Reconstructing and repairing urban models with kinetic data structures

Mulin Yu

UNIVERSITÉ CÔTE D'AZUR

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

THÈSE DE DOCTORAT

Reconstruction et Correction de Modèles Urbains à l'Aide de Structures de Données Cinétiques

Mulin Yu

INRIA Sophia-Antipolis Méditerranée

Présentée en vue de l'obtention du grade de docteur en Informatique
d'Université Côte d'Azur
et de INRIA Sophia Antipolis
Dirigée par : Florent Lafarge

Soutenue le : 2 Décembre 2022

Devant le jury, composé de :
Philippos Mordohai, Rapporteur, Stevens Institute of Technology
Gilles Gesquière, Rapporteur, University Lyon 2
Marc Antonini, Examinateur, I3S-CNRS
Sven Oesau, Examinateur, Geometry Factory
Florent Lafarge, Examinateur, INRIA – Université Côte d'Azur
Bruno Hilaire, Examinateur Invité, GEOFIT
Julien Soula, Examinateur Invité, CSTB

Inria CSTB

Reconstruction et Correction de Modèles Urbains à l'Aide de Structures de Données Cinétiques

Reconstructing and Repairing Urban Models with Kinetic Data Structures

Jury:

Rapporteurs
Philippos Mordohai, Professor, Stevens institute of technology
Gilles Gesquière, Professeur des universités, University Lyon 2

Examinateurs
Marc Antonini, Directeur de recherche, I3S-CNRS
Sven Oesau, Ingénieur de recherche, Geometry Factory
Florent Lafarge, Directeur de recherche, INRIA Sophia-Antipolis

Examinateurs Invité
Bruno Hilaire, Ingénieur de recherche, GEOFIT
Julien Soula, Cadre scientifique, CSTB

Acknowledgements

Thank you to those who have helped me over the past three years.

Résumé

Les modèles numériques 3D compacts et précis de bâtiments sont couramment utilisés par les praticiens pour la visualisation d'environnements existants ou imaginaires, les simulations physiques ou la fabrication d'objets urbains. La génération de tels modèles prêts à l'emploi est cependant un problème difficile. Lorsqu'ils sont créés par des designers, les modèles 3D contiennent généralement des erreurs géométriques dont la correction automatique est un défi scientifique. Lorsqu'ils sont créés à partir de mesures de données, généralement des balayages laser ou des images multivues, la précision et la complexité des modèles produits par les algorithmes de reconstruction existants n'atteignent souvent pas les exigences des praticiens. Dans cette thèse, j'aborde ce problème en proposant deux algorithmes : l'un pour réparer les erreurs géométriques contenues dans les formats spécifiques de modèles de bâtiments, et l'autre pour reconstruire des modèles compacts et précis à partir de nuages de points générés à partir d'un balayage laser ou d'images stéréo multivues. Le composant clé de ces algorithmes repose sur une structure de données de partitionnement d'espace capable de décomposer l'espace en cellules polyédriques de manière naturelle et efficace. Cette structure de données permet à la fois de corriger les erreurs géométriques en réassemblant les facettes de modèles 3D chargés de défauts, et de reconstruire des modèles 3D à partir de nuages de points avec une précision et complexité proche de celles générées par les outils interactifs de Conception Assistée par Ordinateur.

Ma première contribution est un algorithme pour réparer différents types de modèles urbains. Les travaux antérieurs, qui reposent traditionnellement sur une analyse locale et des heuristiques géométriques sur des maillages, sont généralement conçus sur-mesure pour des formats 3D et des objets urbains spécifiques. Nous proposons une méthode plus générale pour traiter différents types de modèles urbains sans réglage fastidieux des paramètres. L'idée clé repose sur la construction d'une structure de données cinétiques qui décompose l'espace 3D en polyèdres en étendant les facettes du modèle d'entrée imparfait. Une telle structure de données permet de reconstruire toutes les relations entre les facettes de manière efficace et robuste. Une fois construites, les cellules de la partition polyédrique sont regroupées par classes

sémantiques pour reconstruire le modèle de sortie corrigé. Je démontre la robustesse et l'efficacité de l'algorithme sur une variété de modèles réels chargés de défauts et montre sa compétitivité par rapport aux techniques traditionnelles de réparation de maillage à partir de données de modélisation des informations du bâtiment (BIM) et de systèmes d'information géographique (SIG).

Ma deuxième contribution est un algorithme de reconstruction inspiré de la méthode Kinetic Shape Reconstruction, qui améliore cette dernière de différentes manières. En particulier, je propose une technique pour détecter des primitives planaires à partir de nuages de points 3D non organisés. Partant d'une configuration initiale, la technique affine à la fois les paramètres du plan continu et l'affectation discrète de points d'entrée à ceux-ci en recherchant une haute fidélité, une grande simplicité et une grande complétude. La solution est trouvée par un mécanisme d'exploration guidé par unefonction énergétique à objectifs multiples. Les transitions dans le grand espace des solutions sont gérées par cinq opérateurs géométriques qui créent, suppriment et modifient les primitives. Je démontre son potentiel, non seulement sur des bâtiments, mais sur une variété de scènes, des formes organiques aux objets fabriqués par l'homme.

Mots clés: Reconstruction de surface, réparation de maillage, reconstruction sémantique, structure de données cinétiques, BIM, GIS, détection de forme planaire, regroupement de nuages de points

Abstract

Compact and accurate digital 3D models of buildings are commonly used by practitioners for the visualization of existing or imaginary environments, the physical simulations or the fabrication of urban objects. Generating such ready-to-use models is however a difficult problem. When created by designers, 3D models usually contain geometric errors whose automatic correction is a scientific challenge. When created from data measurements, typically laser scans or multiview images, the accuracy and complexity of the models produced by existing reconstruction algorithms often do not reach the requirements of the practitioners. In this thesis, I address this problem by proposing two algorithms: one for repairing the geometric errors contained in urban-specific formats of 3D models, and one for reconstructing compact and accurate models from input point clouds generated from laser scanning or multiview stereo imagery. The key component of these algorithms relies upon a space-partitioning data structure able to decompose the space into polyhedral cells in a natural and efficient manner. This data structure is used to both correct geometric errors by reassembling the facets of defect-laden 3D models, and reconstruct concise 3D models from point clouds with a quality that approaches those generated by Computer-Aided-Design interactive tools.

My first contribution is an algorithm to repair different types of urban models. Prior work, which traditionally relies on local analysis and heuristic-based geometric operations on mesh data structures, is typically tailored-made for specific 3D formats and urban objects. We propose a more general method to process different types of urban models without tedious parameter tuning. The key idea lies on the construction of a kinetic data structure that decomposes the 3D space into polyhedra by extending the facets of the imperfect input model. Such a data structure allows us to re-build all the relations between the facets in an efficient and robust manner. Once built, the cells of the polyhedral partition are regrouped by semantic classes to reconstruct the corrected output model. I demonstrate the robustness and efficiency of the algorithm on a variety of real-world defect-laden models and show its competitiveness with respect to traditional mesh repairing techniques from both Building Information Modeling (BIM) and Geographic

Information Systems (GIS) data.

My second contribution is a reconstruction algorithm inspired by the Kinetic Shape Reconstruction method, that improves the later in different ways. In particular, I propose a data fitting technique for detecting planar primitives from unorganized 3D point clouds. Departing from an initial configuration, the technique refines both the continuous plane parameters and the discrete assignment of input points to them by seeking high fidelity, high simplicity and high completeness. The solution is found by an exploration mechanism guided by a multi-objective energy function. The transitions within the large solution space are handled by five geometric operators that create, remove and modify primitives. I demonstrate its potential, not on buildings only, but on a variety of scenes, from organic shapes to man-made objects.

Keywords: Surface reconstruction, Mesh repairing, Semantic reconstruction, Kinetic data structure, BIM, GIS, Planar shape detection, Point cloud clustering

Contents

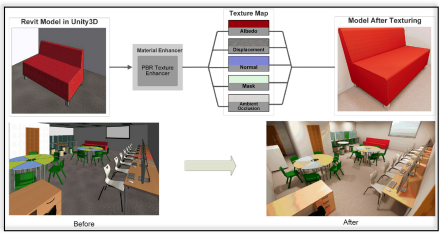
Chapter 1

Introduction

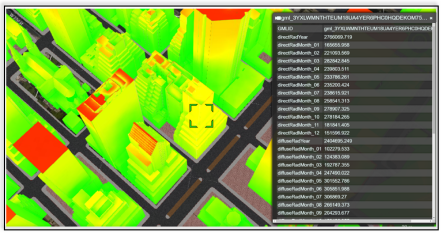
1.1 Context

Urban Digital Twin (UDT) is gaining more and more attention in both the scientific and industrial domains, as a result of the rapid development of big data, artificial intelligence, and the Internet of Things (IoT) [SH20, SHY21]. UDT can be used for intelligent urban management, intelligent urban planning, disaster prevention, disaster control, energy analysis, clash detection and more by constructing a virtual representation of the city [BSL+15], Figure 1.1. The fundamental concept behind a digital twin is creating a virtual model that accurately depicts a real object or scene and is appropriate for the target application scenario. The application scenarios of UDT can be roughly separated into two families: (1) designing and analysing buildings, (2) planning and managing cities. Therefore, Building Information Modeling (BIM) and Geographic Information System (GIS) have been proposed to comprehend and maintain the 3D virtual models in the building and city scales, respectively. In these two systems, the 3D virtual models are formalized in different ways to match the different application requirements. Specially, two most popular formats in these two systems are introduced below and are considered in this thesis.

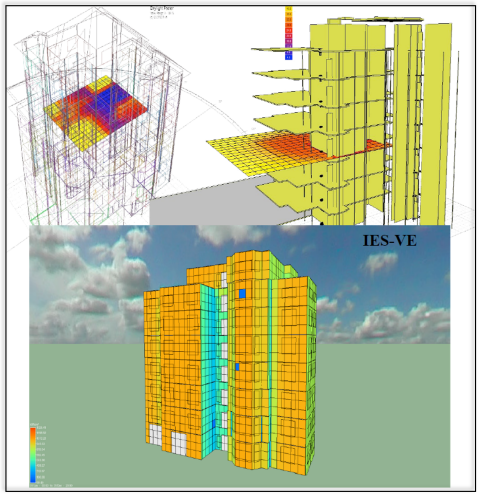
Industry Foundation Classes (IFC). Initiated in 1994, IFC is a CAD exchange data schema designed to describe the context of the building, architecture and construction industry. IFC has been the most widely used format in BIM, which is originally formed with the goal of reducing information loss in the AEC (architecture, engineering, and construction) domains [VSS14]. An IFC model typically describes only one building in detail, Figure 1.2, which will be used during the entire life cycle of the building. The IFC models therefore include additional information, such as project budgets, schedules and organization, that is needed by various vendors and customers



(a) Interactive design

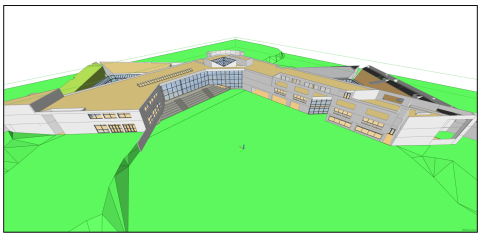


(b) Solar irradiation estimation

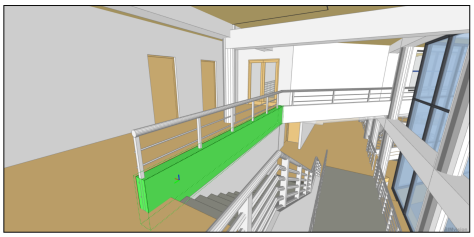


(c) Daylight analysis

Figure 1.1: Visualization of three applications of UDT. BIM-based daylight analysis (c) and interactive design (a). GIS-based solar irradiation estimation (b). Images come from [PMMB22, YNK+18, JJ14].



(a) *Outdoor*



(b) *Indoor*

Figure 1.2: Visualization of the outdoor and indoor scenes of an IFC model. The building is presented based on volumes, a chosen *IfcWall* is shown in green (b). Image visualized by *BIMvision* [Bui].

in addition to geometric and semantic information. As an object-oriented and EXPRESS-based [ISO04] data format, all information in IFC is managed by classes, which contains 130 defined types, 207 enumeration types, 60 select types, 776 entities, 47 functions, and 2 rules in IFC 4 Addendum 2 [OBD+17]. All the classes are subtypes of the root class and can be separated to three types, *IfcObject*, *IfcPropertyDefinition* and *IfcRelationship*. *IfcObject* defines all the physical objects or constructing activities in the AEC context. Attached to the *IfcObject*, *IfcPropertyDefinition* describes all possible attributes as size, owner, name and orientation. The relationship between *IfcObject*s is built by *IfcRelationship*s. Each IFC element, such as

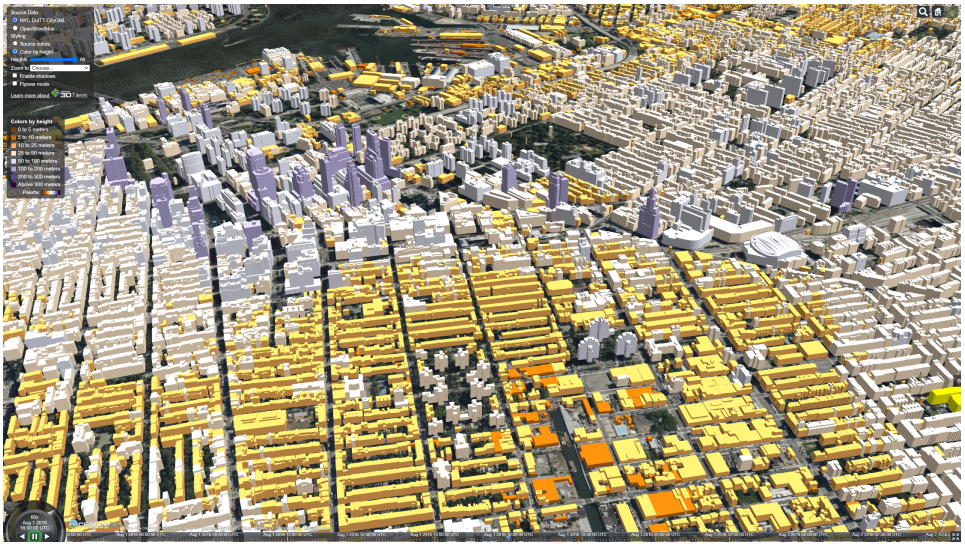


Figure 1.3: A part of New York city LOD2 CityGML model [Dep16] in the Cesium Platform [Ces]. The color is related to the height of the buildings. Each build is represented by its envelop and the indoor part is ignored.

wall, roof and space, is presented as a volume and is associated with a semantic class. There are four geometric presentation paradigms in IFC [OBD+17], primitive instancing, boundary representation (B-rep), constructive solid geometry (CSG), and sweep volume:

1. Primitive instancing: An element is represented based on several predefined primitives such as simple shapes.

2. B-rep: As a most straightforward way, elements are represented by the set of their boundary surfaces.

3. CSG: Series of Boolean operations (difference, union and intersection) of a set of primitives are used to present an element. The primitives can be simple shapes like spheres, cones and cylinders.

4. Sweep volume: An element is implicitly presenting with a 2D shape and a curve. The curve is the trajectory of the 2D shape.

To be used in the downstream applications, each element should be represented by a geometric and topological valid volume, and there should not exist gaps or overlaps between any pairs of adjacent elements.

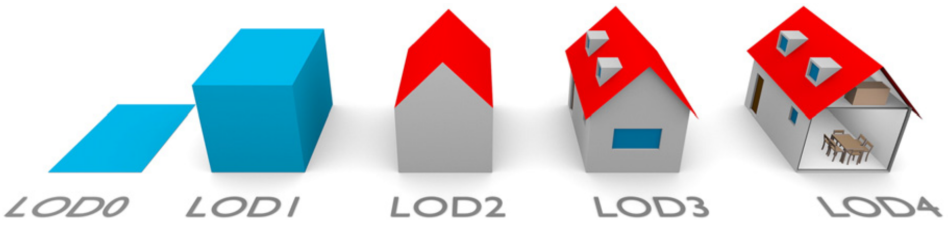


Figure 1.4: The five LODs of CityGML 2.0. From LOD0 to LOD4, the virtual models are more and more precise. They are employed in various application contexts. Image taken form [BLS16]

CityGML. Initiated in 2002 by the Special Interest Group 3D, CityGML [GL12] is the most prominent standard formalism of Geographic information system (GIS) [Cha16] for storing, managing, analyzing and visualizing geographic data. The geographic information provided in CityGML format relates to positions on the surface of the Earth, such as streets, buildings, and vegetation. In addition to the geometry of urban objects, CityGML can also record the topology, semantics and appearance of urban objects. Specifically, CityGML can present an object in different levels of detail (LODs). From LOD0 to LOD4, more and more details are presented as illustrated in Figure 1.4. LOD2 CityGML models are typically used at the city scale to plan urban areas, navigate outdoor pedestrians, simulate the environment, and manage facilities. As a result, we only consider the LOD2 CityGML models in this thesis. According to ISO 19107 standard [ISO03], the surfaces and curves in CityGML can only be planar and linear, respectively. The non-linear structure such as the roofs of nymphaeum should be approximated by piecewise planar surfaces. The geometric objects are presented by boundary representation, e.g. buildings are preferred to be presented by watertight and manifold solids that are composed by their bounding surfaces. Even though a soup of unstructured surfaces is also permitted to present an object, this representation is not friendly to some downstream applications that require geometric processing on meshes. Semantic information is associated with each surface and is enriched as the LOD increases. For instance, the surfaces of a building in LOD1 can only contain *building* semantic information, while they can be identified as *roof*, *wall* and *ground* in LOD2. A CityGML file can be created by grouping together various urban digital twin components, such as buildings, bridges and tunnels, Figure 1.3.

Both IFC and CityGML portray the real world in a compact manner, despite the fact that they are two completely distinct forms. In other words, the 3D virtual models are displayed by few facets that have big areas. By compactly formalized, the 3D virtual models can be stored in a small amount of memory, which make the processing programs effective. In addition, compact representation has much simpler and clearer topological structure, which makes it simple for users to edit them.

Typically, the 3D virtual models are generated through two different ways: (1) Manually: 3D virtual models are designed and created by architects or designers using specific software such as Revit. (2) Automatically: they can be reconstructed from data measurements such as 3D point cloud using reconstruction methods. Even though manual methods are the dominant technology for generating 3D virtual models, some topological, geometric and semantic errors can be introduced into the 3D virtual models due to the limitations of modeling software and the lack of guidelines. Such errors are catastrophic for certain applications like the topological errors can collapse the energy analysis. As a result, correcting defect-laden 3D virtual models before putting them into the downstream applications is often necessary processing step. In the domains of computer vision and computer graphics, reconstructing virtual models from data measurements in an automatic way is a long-standing problem. There is still a long way to go before automatically reconstructing semantic enriched IFC or CityGML models from data measurements. Therefore, researchers attempt to first reconstruct compact meshes from data measurements.

1.2 Challenges

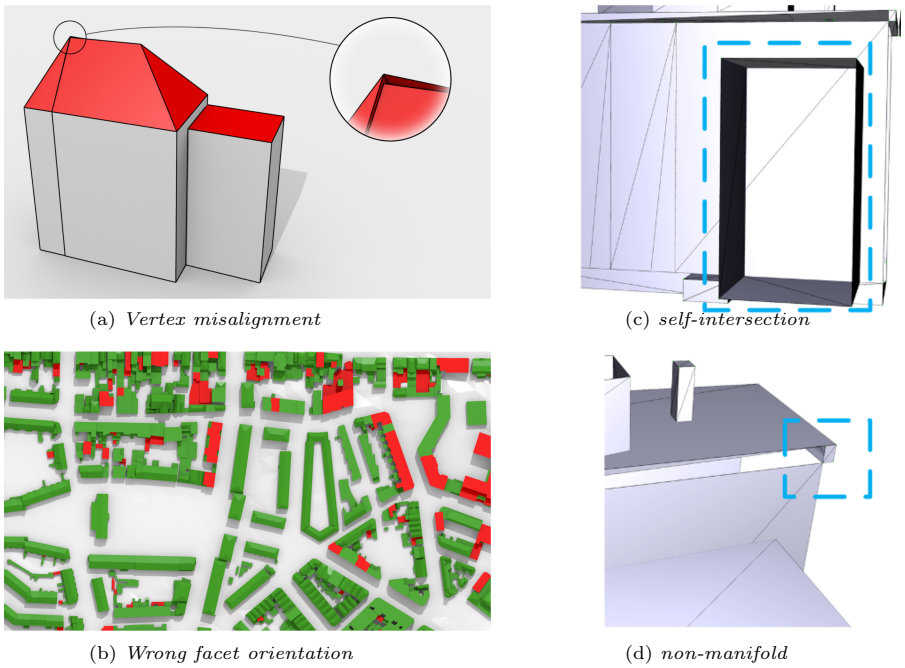
The ultimate objective in the field is to automatically generate ready-to-use models from readily accessible defect-laden 3D models or point clouds. 3D model repairing and compact mesh reconstruction are difficult problems, mainly due to the variety of input and the ambiguity on quality evaluation.

1.2.1 Repairing

The challenge of repairing defect-laden 3D models comes from three primary sources: the variety of input, the variety and the ambiguity of errors.

Variety of input. As previously mentioned, 3D virtual models are saved in different formalisms depending on different application scenarios. Specially, the LOD2 CityGML models ignore the thickness of building elements, i.e. buildings are represented by their envelopes. In contrast, IFC models are used to thoroughly describe a certain building and take element thickness into account. As a result, the repairing methods should be general enough to handle both the surface-based and volume-based representations.

Variety of geometric and topological errors. Urban models frequently contain many topological and geometrical errors [BLD+16], Figure 1.5. These errors are not independent, for instance, a misaligned vertex may belong to a self-intersecting facet. One major drawback of traditional repairing methods



(a) *Vertex misalignment* (c) *self-intersection*

(b) *Wrong facet orientation* (d) *non-manifold*

Figure 1.5: Visualization of four kinds of geometric and topological errors in urban models. (a) shows the vertex misalignment error, which is frequently introduced during conversion and storage. The facets in LOD2 CityGML models are typically associated with orientation that points outward, the red facets are associated with the wrong orientation in (b). (c) and (d) represent the common topological errors: self-intersection and non-manifold. Image taken form [BLD+16, ZLSF18].

is that extra errors may be introduced into the output models since they iteratively conduct predefined local repairing operations based on the assumption that errors are independent. Therefore, the ideal repairing method should be robust enough to deal with all the errors and correct the errors in a global way that guarantees the validity of the output models.

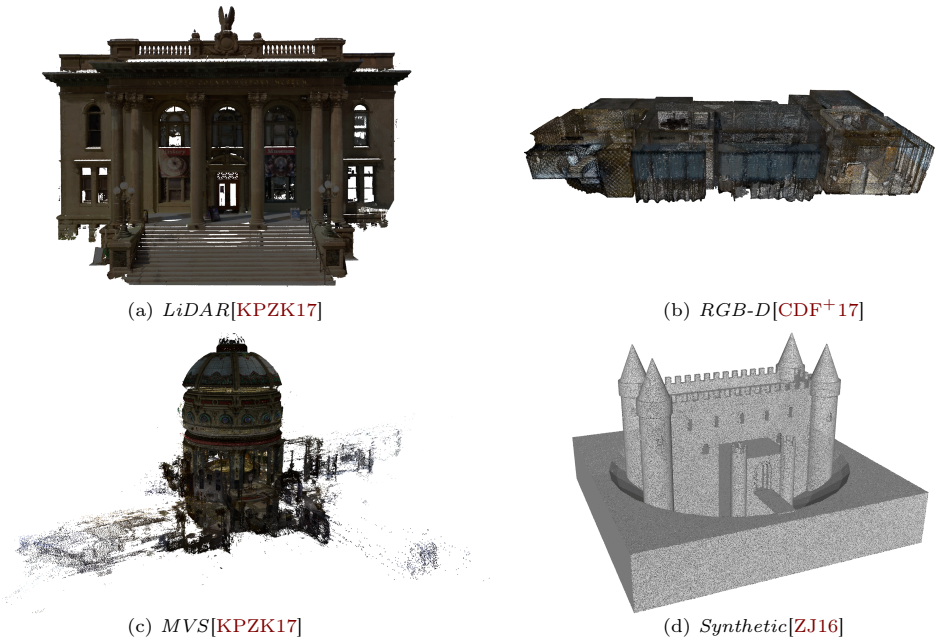
Ambiguity of semantic errors. Semantic errors are sometimes subject to interpretation. For instance, it is hard to automatically determine the semantic class for the common part of two intersecting *IfcElement*s. As a result, the ideal repairing methods should have the ability to highlight these areas where the semantic classes are ambiguous and enable the experts to quickly locate and correct the semantic errors.

1.2.2 Reconstruction

Compact mesh reconstruction is a long-standing topic in the computer vision and computer graphics fields. The two main challenges are the variety of input and the quality metrics. Note that we only consider point clouds as data measurements in this thesis.

Variety of input. The point cloud is the most common data measurement in the urban context. It is a collection of unstructured points in space that seeks to present an object or a scene realistically. The point clouds obtained differ in terms of the type of physical thing described and the types of acquisition tool. On the one hand, the styles of structures and indoor scenes vary greatly in a big urban context. The reconstruction methods should have the ability to deal with different types of objects and scenes. For instance, several existing reconstructing methods heavily rely on geometric hypotheses like the Manhattan World assumption and the regularity of objects, which can not generally perform in all situations. On the other hand, point clouds can be gathered and created using different tools:

1. **Light Detection And Ranging (LiDAR).** LiDAR is a laser-based remote sensing technology. The core principle of LiDAR is the reflection of light. Similar to radar, however, LiDAR employs lasers instead of radio waves. It first emits a laser pulse to the surface of an object, then catches the reflected laser and finally the location of reflection



(a) *LiDAR*[KPZK17] (b) *RGB-D*[CDF+17]

(c) *MVS*[KPZK17] (d) *Synthetic*[ZJ16]

Figure 1.6: Visualization of four kinds of 3D point clouds. Point clouds may have other attributes besides coordinates, such as color in (a), (b) and (c).

can be determined. To achieve laser emitting, laser capture and positioning, a LiDAR scanner should have a laser emitter, a photodetector and an embedded signal processing chain. Depending on different applications and environments, the wavelengths of beams emitted by the laser emitter should be adjusted. The wavelengths vary between 250 nanometers to 10 micrometers. And the resolution of LiDAR data is determined by the pulse width, a shorter pulse can lead to high resolution. There are two platforms for LiDAR: static LiDAR and dynamic LiDAR. The former employs a static device that scans in a ball pattern. The latter employs one or more devices on cars or planes that scan in a circular line pattern.

2. **RGB-D cameras.** RGB-D cameras have drawn a lot of interest in recent years due to their affordable cost and are widely applied in face unlocking, 3D reconstruction, augmented reality and Simultaneous Mapping And Localization (SLAM). RGB-D cameras measure the depth of each pixel on the basis of conventional RGB cameras. The depth is typically measured by using structured light: an infrared light pattern is first emitted by an infrared laser projector, which is then cap-

tured by one or more infrared cameras, finally, the depth is obtained by comparing the deformed pattern to reference patterns that are previously recorded at known depths and stored in the device. The effective measuring distance ranges from 10cm to 10m, however, the accuracy of the acquired depth degrades as the measuring distance is extended. As a result, RGB-D cameras are better suited to indoor scenes than large outdoor scenes. Additionally, benefiting from the properties of structured light, RGB-D cameras are unaffected by ambient lighting conditions and the texture of the objects.

3. **Multiple View Stereo (MVS).** MVS is typically used to extract 3D point cloud from a set of covered images in computer vision. This technique aims at building a bridge between 2D to 3D. As a pipeline, Structure-from-motion (SfM) is first inducted to calculate the camera parameters for each image by minimizing camera projection error on feature points. Then a dense point cloud is reconstructed by finding the 3D coordinate for almost every pixel. MVS can be used in different scales of scenes: indoor scenes, small-scale outdoor scenes and large-scale outdoor scenes. Recently, some deep learning methods [CHXS20, GFZ+20, YLL+18] have also been proposed to reconstruct point clouds from multi-view images.

Recently, deep learning has been also widely used for point cloud processing such as: reconstruction and shape detection. Due to the nature of supervised learning, a large amount of annotated point clouds are required to train neural networks. The manual annotation of the acquired real-world point clouds is very time-consuming and laborious work. To simplify the manufacturing process of the training dataset, semantically enriched point clouds are usually synthesized from existing virtual models, like CAD models [KMJ+19]. Uniform sampling is the most common manner. To make the synthetic point clouds closer to the real-world ones, the sampled point clouds are typically intentionally injected with noise. In the meanwhile, there exists software [GKUP11] that can simulate LiDAR scanning point clouds from a mesh. Figure 1.6 illustrates the three real-world point clouds and a synthetic point cloud and Table 1.1 lists their characteristics. High-quality point clouds are difficult to obtain on a limited budget. Therefore, the reconstruction methods should be robust enough to handle easily available and poor-quality point clouds, such as sparse, noisy and non-uniform MVS point

		Noise	Outliers	Range	Uniformity	Costing
Point clouds	LiDAR	Low	Low	High	High	High
	RGB-D	High	High	Low	Low	Fair
	MVS	High	High	Fair	Low	Low
	Synthetic	Any	Any	Any	Any	Low

Table 1.1: Evaluation of the characteristics of different types of point clouds.

clouds. In addition, the size of obtained point clouds are typically quite large due to the characteristics of urban scenes no matter the acquisition tools. The reconstruction methods should be scalable to handle input with numerous elements.

Quality metrics. There is not a single, universally accepted metric to evaluate the quality of the reconstruction methods. Different criteria might be taken into account in various application contexts. For instance, while SLAM always overlooks details like the texture of walls in order to reconstruct a scene in real-time, heritage recording attempts to reconstruct as many details of a scene or an object as possible. Therefore, it is obviously unconscionable to consider only one criterion. A broad measure that incorporates numerous criteria should be proposed. Specially, the considered measurements are always conflict with each other, e.g. it is unable to simultaneously achieve high accuracy and high efficiency with limited resources. As a result, the broad measure is a trade off between conflict criteria, which requires the reconstruction methods to have the capacity to trade off between these criteria.

1.3 Outline

In this thesis, we investigate the 3D model repairing and the compact mesh reconstruction. The structure of this thesis is the following:

1. Chapter 2 covers the previous work related to the problems.
2. Chapter 3 describes the limitations of existing methods and lists our contributions.
3. Chapter 4 presents a method for repairing geometric errors in 3D urban models with kinetic data structures.
4. Chapter 5 presents the details of our contribution to compact mesh reconstruction: detecting good planar primitive configurations from unorganized point clouds, and a practical reconstruction pipeline that combines the explicit kinetic framework and implicit neural network.
5. Chapter 6 draws the conclusion and perspectives of our work.

Chapter 2

Literature Review

We review previous works related to two aspects of the thesis: repairing defect-laden 3D urban models and compact mesh reconstruction.

2.1 Repairing 3D models

In this section, we review the repairing methods for 3D models. The traditional mesh repairing methods are first reviewed before the specific ones that can be used to urban models (CityGML and IFC).

2.1.1 Mesh repairing

Our problem is a specific instance of the mesh repairing issue whose goal is to correct geometric errors contained in urban models. Deeply studied in Geometry Processing [CAK12, PSMA16], we separate the mesh repairing methods into two families: local methods and global methods.

Local methods. In a local manner, several specially designed local operators are utilized in a small region in the vicinity of the individual flaw and defects that are located by traversing the input defected mesh, Figure 2.1. Such techniques are local and assume that geometric errors are relatively isolated from each other [GTLH01, BK97, Att10, BS95, MW99, WLL+12]. These methods can efficiently handle straightforward situations, however, they typically fail on complex cases, such as when several errors are dependent. Therefore, they are always employed in some special contexts to address predictable errors.

Global methods. Global methods correct the errors by globally re-meshing the defect-laden meshes. It is common to need an intermediate data structure, such as the constrained Delaunay triangulation or the voxel grid. For

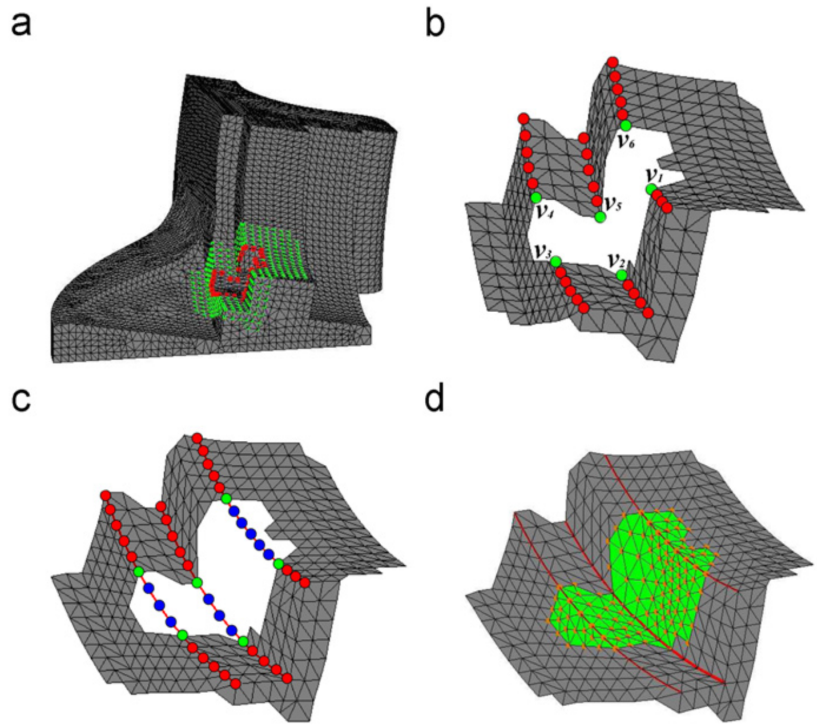


Figure 2.1: Overview of a local method [WLL+12]. A hole in the input mesh is first located (a), then the feature vertices are selected and the boundary feature vertices are marked in green (b). The feature curves are reconstructed and some blue vertices are sampled on the curves located in the hole (c). The sampled points are used to reconstruct the missing mesh patch to fill the hole (d). Image taken from [WLL+12].

instance, [NT03] proposes a voxel-based method with morphological operators to handle holes, double walls and intersecting parts. [GDJY19] leverages constrained Delaunay triangulation to re-mesh each CAD patch in a 2D parametric domain before projected back to the 3D space. [MPR12] builds a discrete parametrization with an orthogonal gradient method on radial basis functions before remeshing the surface in the parametric space with a computed inverse mapping. Even though global methods are much more robust and can handle more complex configurations than local methods, they typically modify the entire input mesh and can smooth out some sharp features such as corners and creases. In addition, it usually needs more computing resources than local methods.

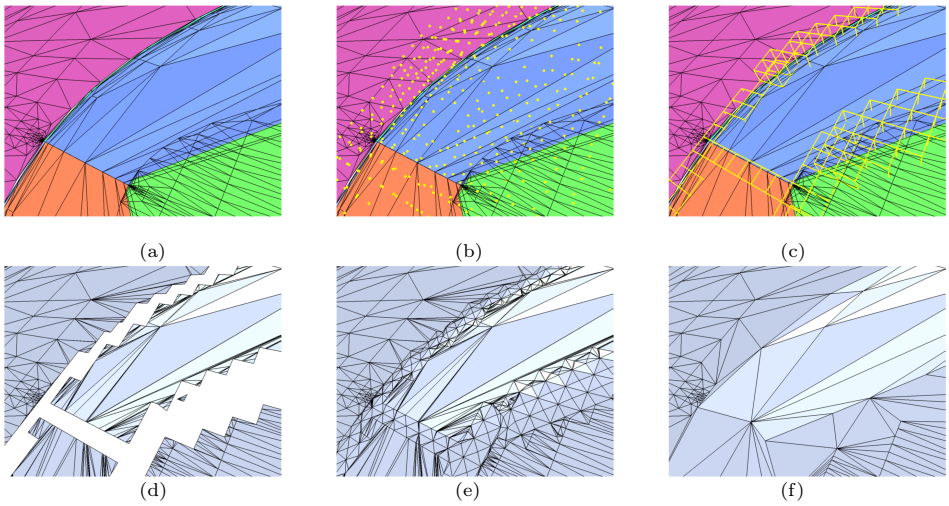


Figure 2.2: Pipeline of a local re-meshing method [BK05]. From the input CAD patches (a), the set of critical vertices in a local neighborhood around the defects are first determined (b). Then these critical vertices are converted into a set of voxels (c). The input CAD mesh is then clipped by the set of voxels (d) and filled by the reconstructed mesh inside the voxels (e). Finally, the mesh is simplified (f). Image taken from [BK05].

Specially, some methods try to re-mesh the region in vicinity of the flaw and defects, i.e. a local re-meshing method. For instance, [BK05] relies on a snapping process in the vicinity of intersections and cracks to re-mesh inconsistencies in a manifold way, Figure 2.2, however, this method can not fill holes in the mesh. The mentioned global methods are relatively efficient to repair geometrical and topological errors but do not allow semantic errors contained in BIM and GIS data to be corrected.

2.1.2 Repairing of CityGML and IFC data

Some works study the validity of CityGML and IFC models and propose specific methods for repairing them. Such tasks are delicate as the errors contained in CityGML and IFC models are numerous and typically emerge from geometric, topological and semantic approximations [BLD+16, OBD+17]. [KNO+20] proposes a set of validation algorithms to test the relationships between geometries in IFC models. [LGR15] detects the frequented geometric error in IFC models: clash errors, space definition errors and surface orientation errors, and semi-automatically corrects these detected errors by

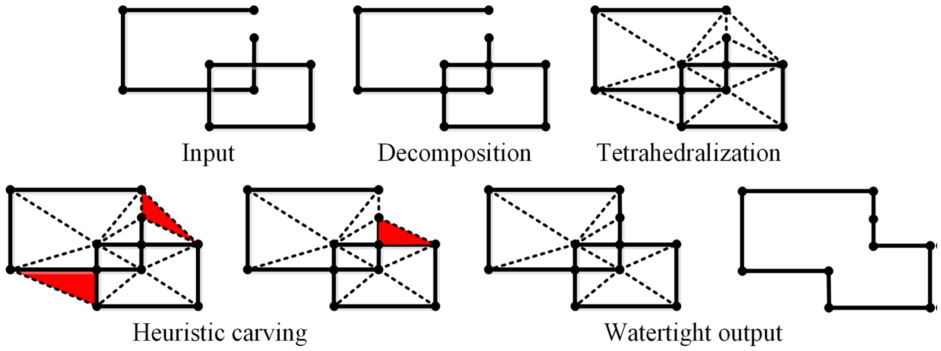


Figure 2.3: Work flow of a LOD2 CityGML model repairing method [ZLSF18]. A constrained tetrahedralization method is first used to build a wrap that embeds the decomposed input facets. The carving strategies are then used to shrink the wrap under topological and geometric constraints. Image taken from [ZLSF18].

local operations. [AWW+14] proposes a set of geometric rules to test the validity of CityGML models and local operators to repair certain types of errors. [ZSL14] exploits a similar strategy with an efficient recursive repairing framework. [DDVM14] and [ZLSF18] entirely re-build the input models using a space partitioning data structure. The former proposes an automatic method to convert a topology-free model into a cellular decomposition using combinatorial maps whereas the latter uses a heuristic shrink-wrapping algorithm for reconstructing valid solid-based LOD2 buildings through a constrained tetrahedralization, Figure 2.3. As explained in [AWW+14], these algorithms typically rely on combinations of heuristics that make them efficient for a specific type of input model and specific types of errors only.

2.2 Compact mesh reconstruction

We distinguish two pipelines for reconstructing compact meshes from point clouds: (1) Simplification methods: Simplifying dense meshes that are reconstructed from 3D point clouds. (2) Shape assembling methods: Assembling shapes that are detected from 3D point clouds.

2.2.1 Simplification methods

As the most straightforward strategy, simplification methods first estimate a smooth implicit function and generate a dense mesh from it, the dense mesh then is simplified into a compact mesh.

2.2.1.1 Dense meshes from point clouds

Dense meshes are typically generated by first estimating an implicit function from the input points, such as the signed distance function, unsigned distance function and occupancy function, and then by extracting the isosurface using the methods such as Marching cubes [LC87]. As the most widely used implicit reconstruction method, the Poisson Surface Reconstruction method [KBH06] and its extension [KH13, ZAB+21] estimate the smoothed indicator function based on discretization such as tetrahedralization and octree. Another implicit reconstruction method SSD [CT11] estimates the signed distance function. Despite the fact that these techniques are frequently utilized in practical applications, some extra attributes, such as oriented normals, are necessary.

Some implicit neural networks are proposed to reconstruct 3D models from point clouds due to the quick development of deep learning techniques and the watertight and no self-intersection guarantees of implicit functions. Moving Least-Squares function [LGP+21], signed distance function [JSM+20, EGO+20, PFS+19], unsigned distance function [CMPM20] and occupancy function [BM22, CAPM20, MON+19, PNM+20] are estimated by neural network. Among them, the estimated distance functions are typically truncated to make the network concentrate on the space around the isosurface. Given a query point, the implicit neural network can output a value to indicate its corresponding position in relation to the isosurface. Note that for most neural networks, learning the implicit function just requires the coordinates of points. This enables them to be employed even when the additional attribution, like oriented normals, is difficult to collect.

2.2.1.2 Simplification of dense meshes

Once the dense mesh is reconstructed, it is simplified to a compact mesh with much fewer facets [LRC+03]. A first way for simplifying the dense meshes is iteratively replacing one edge with a single vertex (merging two connected

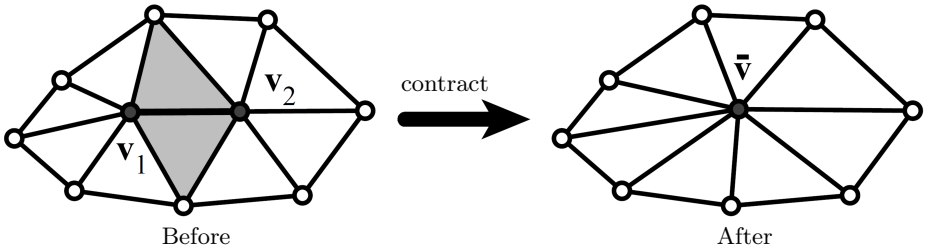


Figure 2.4: Edge contraction (collapse). Edge v_1v_2 is replaced by a single vertex $\overline{v}$. Image from [GH97].

vertex to one) based on different cost metrics. This removes one vertex, three edges, and two facets, in Figure 2.4. In this line of methods, [GH97] first proposes the Quadric Error Metrics (QEM) to gauge the distance between the new vertex and the original mesh. The QEM is used to organize the ordering of edge collapse as well as to determine the location of the new vertex produced by collapsing edges. One year after, [LT98] seeks to maintain the original model volume and the surface area near boundaries instead of considering the distance between the new vertex and the original mesh. To deal with large scale models, [Lin00] presents a way to simplify large polygonal models by combining clustering and QEM. In addition, it proposes a way to locate the "best" new vertex in degenerated case where the collapsed facets are nearly on the same plane. In addition to taking geometric error into account, [Hop99] presents a new technique for efficiently and accurately simplifying mesh with attribute data such as normals and color by proposing a new quadric error metric based on geometric correspondence in 3D. To better preserve the structure of objects or scenes, [SLA15] analyzes both local error metrics and global (structure-aware) error metrics by introducing planar proxies. It is suitable for objects or scenes that contain a large amount of planes. To make the simplification method more tolerant and produce more uniform triangulations, [TK20] introduces probabilistic quadrics that can be robustly minimized by solving a straightforward linear problem. These QEM-based mesh decimation methods are efficient, however, the simplified facets on the planar parts of an object can not be perfectly co-planar.

Another way for simplifying dense meshes consists in using planar polygons to approximate the clustered tiny facets, shown in Figure 2.5. [CSAD04] partitions the dense mesh by minimizing an error function between facets of an input mesh and planar proxies. The error function is constructed using

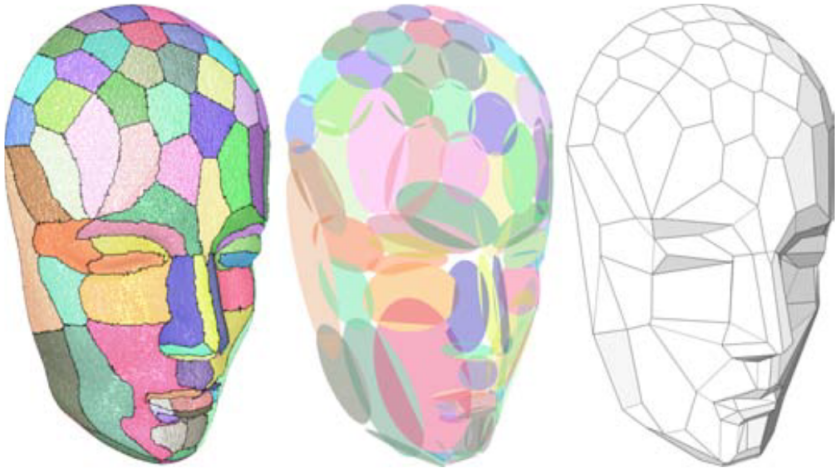


Figure 2.5: Dense mesh partitioning and approximation. The input dense mesh is divided by error-driven partitioning (left). And a planar proxy is extracted for each cluster (middle). The proxies are then used to generate an approximated compact mesh (right). Image sourced from [CSAD04].

the L^2 distance or normal deviation and minimized by Lloyd's algorithm. For fabricating multiplanar models, [CSaLM13] iteratively assigns one plane to each facet by minimizing a cost and deforms the mesh towards planar segments. These methods are efficient on clean input that is both geometrically and topologically accurate, which makes them only deliver good results on synthetic data in practice. Furthermore, these approaches lack flexibility since users must predefine the number of planar proxies.

2.2.2 Shape assembling methods

The other family of methods for reconstructing compact meshes from point clouds consists in detecting planar shapes and assembling them into watertight and manifold compact meshes. The detected planar shapes are typically approximated from sets of clustered points. They act as an intermediate presentation between the input points and the reconstructed compact meshes. As a pipeline, the quality of the detected planar shapes can directly impact the quality of output compact meshes. In accordance with the ordering of the pipeline, we first review the existing planar shape detection methods before moving on the shape assembling methods.

2.2.2.1 Planar shape detection

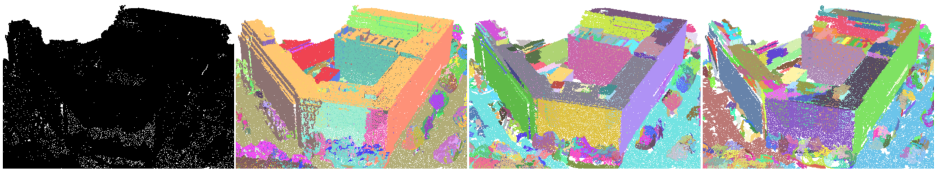


Figure 2.6: Visualization of configurations of planar shapes. Given an input point cloud (a), RANSAC [SWK07] may obtain increasingly precise planar shape configurations from (b) to (d) by gradually reducing the fitting tolerance. Image taken from [OVJ+21].

Detecting planar primitives from 3D point clouds is a long-standing problem, it consists in grouping input points into clusters while associating a parametric plane to each of them. User-specified parameters typically can be fine-tuned to control the precision of the output configuration, as shown in Figure 2.6. We distinguish four families of methods for fitting planar primitives to unorganized 3D point clouds. Note that most of these methods can also detect quadric surface primitives.

Incremental mechanisms (IM). RANSAC and Region Growing are two widely used mechanisms that detect primitives in an iterative manner. The former [SWK07, RCP+13, SM19] samples many plane hypotheses and verifies them against the input data. The plane hypothesis with the largest number of inliers is then kept as a primitive. The latter [MLM01, RVDHV06, VTHLB15] operates by growing a local plane hypothesis in a spatial neighborhood of a seed point. Voting schemes in discretized spaces of the primitive parameters [BELN11, CC08, DI15, SSBC20] are also a popular strategy when the input points are accurately oriented. While Gaussian sphere mapping is the traditional choice for planes and cylinders [QZN14], more complex parameter spaces can also be used for fitting any type of quadrics [BBN+20]. These fast and scalable mechanisms constitute the de facto solutions from real-world data with efficient implementations in popular geometry processing libraries such as CGAL [OVJ+21]. However, they do not control well the output quality, leading often to overly complex plane configurations.

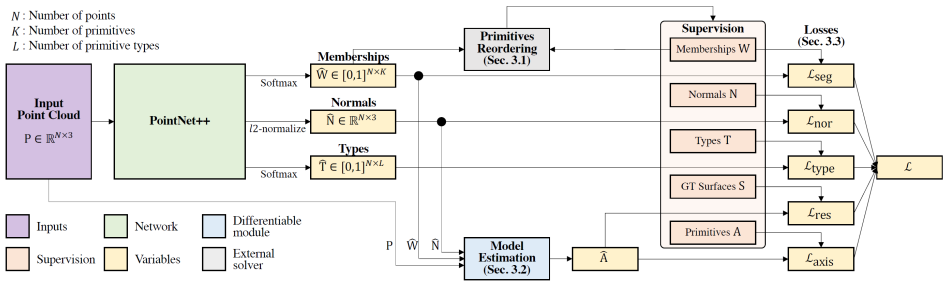


Figure 2.7: Network architecture of a deep learning shape detection method [Li,19]. From a set of input points, PointNet++ is first used to extract three per-point properties: Memberships, Normal and Types. The order of ground truth primitives are matched with the output in the primitive reordering step and the primitive parameters are estimated from the point properties in the model estimation step. Image taken from [Li,19].

Energy-based models (EB). Variational shape approximation [CSAD04, WK05] minimizes an approximation error between the input data and a set of primitives using Lloyd's clustering algorithm [Llo82]. This approach can be combined with sequences of splitting and merging of primitives so that the number of primitives has not to be constant and fixed a priori by the user [SZP20]. Input data are however supposed to be free of outliers, leading to good results on synthetic data only. Another popular strategy consists in detecting a set of plane hypotheses before assigning one of them to each input point using a multi-labeling energy minimization [IB12, PCYS12, PERW16]. While a priori knowledge on label smoothness, output complexity or geometric regularity can be encoded easily, such discrete formulations require good plane hypotheses. This can rarely be guaranteed in practice with existing incremental mechanisms, even when plane hypotheses are enriched with geometric and spatial priors [MMBM15]. Such methods do not formulate energies in the whole solution space (a mixed discrete-and-continuous configuration space) where point assignment and estimation of plane parameters are operated jointly.

Neural network architectures (NN). Deep learning methods have recently emerged as a promising solution to the tedious parameter tuning problem of traditional algorithms. The end-to-end networks SPFN [Li,19], Figure 2.7, and ParSeNet [SLK+20] predict per-point properties using off-

the-shelf architectures, typically Pointnet++ [QYSG17], before estimating the primitive types and parameters. CPFN [LSC+21] proposes an adaptive patch sampling network to assemble primitive detection results at coarse and fine scales. HPNet [YYM+21] extracts primitives using a mean-shift clustering operating on a combination of three learned features with adaptive weights. These networks require high computing resources and can only handle point clouds of a hundred thousand points at best. PrimitiveNet [HZS21] offers a solution to this scalability issue with a region growing mechanism operated from the output of two networks, one producing per-point high dimensional features and the other predicting whether two neighboring points belong to the same primitive. This solution, however, requires defect-free, dense meshes as input. In practice, these learning methods do not generalize well on real-world data as training sets are composed of synthetic point clouds sampled from datasets of Computer-Aided Design (CAD) models [RRQ+21, KMJ+19, ZJ16]. Note that some neural networks also detect planes from single image [YZ18, LKG+19, QF20] or depth maps [ZYY+17].

Methods with regularity enforcement (RE). Some methods are designed to enforce geometric regularities between planes such as parallelism, orthogonality or some types of symmetry. This can be done by alternating fitting and regularization of primitives within the traditional incremental mechanisms, i.e. Region Growing [OLA16] and RANSAC [LWC+11], or by inserting pairwise priors in energy-based models [MMBM15]. Mutually orthogonal planes that respect the Manhattan-World assumption [CY00] can also be fitted efficiently by Gaussian sphere mapping [SRF+14]. Some methods [FLD18, LMBM20] go further by analyzing the structure of objects at different key abstraction levels so that no user-specified fitting tolerance is required. In practice, the assumptions for such geometric and structural regularities are relevant for specific application domains only.

After going over the four different types of planar shape detection methods, we list their advantage and disadvantage in Table 2.1. None of them are able to operate in the whole (mixed continuous and discrete) configuration space of the problem.

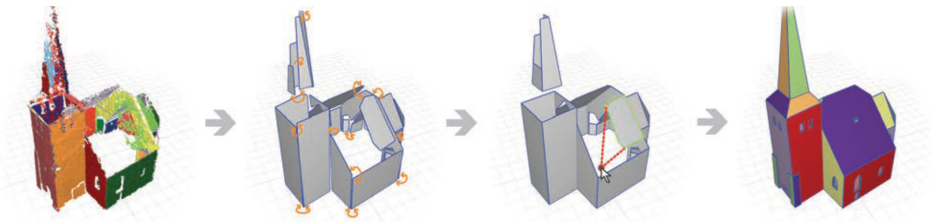
IM	Pros	Fast, Scalable and efficient implementations *(CGAL)*.
	Cons	Poor control of output quality (overly complex).
EB	Pros	Fast.
	Cons	Can not explore the whole solution space.
NN	Pros	No parameter setting.
	Cons	Big dataset, not scalable, not robust on real-world data.
RE	Pros	Working well on specific application.
	Cons	Not generic and inheriting cons from **IM** and **EB**.

Table 2.1: Pros and cons of four families of planar shape detection methods.

2.2.2.2 Shape assembling

We broadly categorize shape assembling methods into two families: connect ivity-based methods and partitioning-based methods. Both of them aim to reconstruct a compact mesh from the detected planar shapes.

Connectivity-based methods. Connectivity-based methods extract the vertices, edges, and facets of the resultant compact mesh by analyzing the adjacent relationships between detected planar shapes. [VKVLV11] proposes a guided α-shape that is bounded by a collection of intersection lines between planes, which makes the soup of planar shapes that can be connected along the intersection lines. However, this method is reliant on the guidelines and cannot handle non-uniform distributed and missing data. To handle scanned sparse building point clouds that partially or completely miss faces, [CC08] first finds all the intersected lines between detected neighbor planar



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

Figure 2.8: Pipeline of a semi-automatic method [ASF+13]. Departing from a (a) segmented point cloud, local adjacency relations are automatically discovered (b) and used to snap the polygons (c). Finally, a compact mesh is reconstructed with the help of user interaction (d). Image from [ASF+13].

shapes and the boundaries that may infer missing facets, and then extracts the target compact mesh with the aid of cluster graphs and user interaction. However, the neighboring information is determined erroneously by the minimum distance between the corresponding regions of points, which may introduce wrong adjacent relation. A semi-automatic method [ASF+13] devises five mechanisms to constrain the permitted adjacencies before aligning all the detected polygons, Figure 2.8. It also proposes interactive editing with simple 2D operators to handle missing data. Instead of user interaction, [LA13] first performs a 3D constrained Delaunay triangulation on a structured point set that is consolidated using the detected planar shape, then the output mesh is automatically extracted using graph cut. Optionally, the output mesh can be simplified based on the classified points without loss of accuracy, however, it is not suited to missing data.

Partitioning-based methods. To create a compact mesh, partitioning-based methods first divide the 3D space (inside the bounding box of the input point cloud) into polyhedrons departing from the detected planar shapes. Compared to connectivity-based methods, partitioning-based methods are global and robust to challenging data, e.g. missing data. Several works are based on a constrained Delaunay triangulation [VKVLV13] or a voxel grid [SDK09], however, they can not directly generate compact meshes and some simplification methods are required as post-procession. Compared to these dense 3D partitioning, we prefer more sparse partitioning which embeds the detected planar primitives like (d) shown in Figure 2.9. [CLP10] uses a two-level hierarchy to partition the 3D space into a polyhedral cell complex, Figure 2.9. The strategy of introducing ghost planar primitives and the heuristic for sorting the inserted primitives, however, are not general and are based on specific assumptions. As a result, [BdLGM14] uses a planar arrangement rather than a two-level hierarchy such that the resulting polyhedral cell complex is independent of the planar primitive insertion order. Additionally, it introduces a more reasonable strategy to generate ghost planes for *occluding segments* only. However, building a planar arrangement is time consuming, which makes the method not scalable. To build the polyhedral cell complex more efficiently, [OLA14] transforms the 3D partitioning into a set of 2D partitioning by separating the 3D space into horizontal slices. However, this method is only valid for multi-floor indoor scenes. Filtering

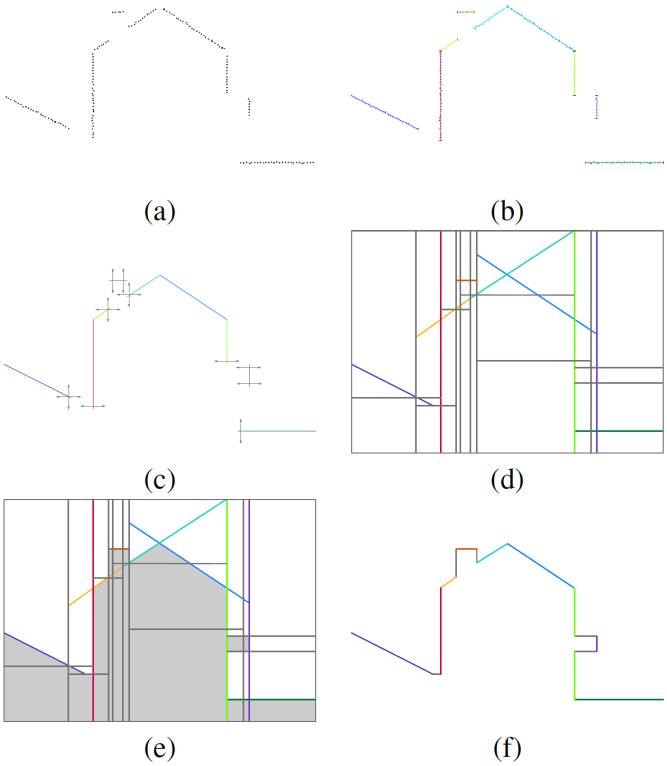
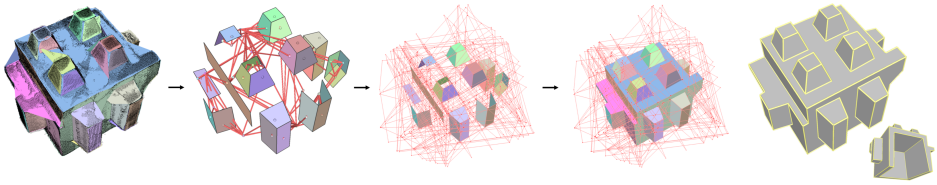


Figure 2.9: Overview of a partitioning-based method [CLP10]. Departing from a point cloud (a), planar primitives (b) are detected and ghost planar primitives (c) are introduced. The 3D domain is then sliced into a set of convex polyhedrons (d) by inserting the planar primitives in order. The final compact mesh (f) is extracted by labeling the occupancy of each polyhedral cell. Image taken from [CLP10].

and simplifying the detected planar primitives is another way to reduce the computation burden [NW17]. This method is not scalable and cannot handle more than a hundred planar primitives. Furthermore, [BL20] proposes a more reasonable way to divide the 3D space, which extends all detected planar shapes at the same rate until they collide with one another and create a compact polyhedral partition. It significantly reduces the number of polyhedrons compared to the planar arrangement and improves the effectiveness of partitioning the 3D space. Recently, [CTZ20] leverages the Binary Space Partitioning (BSP) tree to facilitate 3D geometry learning. It progressively predicts the planes, convex shapes and combined convex shapes through a



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

Figure 2.10: Overview of an assembling method that combines connectivity and partitioning [FL20]. Departing from the detected planar primitives (a), connectivity analysis allows to extract the structurally-valid facets (b). The partition is built by slicing the spatially-close unprocessed primitives while embedding the structurally-valid facets (c). The facets of the reconstructed compact mesh (d) are extracted by minimizing an energy using an integer programming solver (e). Image taken from [FL20].

neural network. Although it is a brilliant integration between compact mesh reconstruction and deep learning, the number of planes and convex shapes should be predefined by users, which decreases the resolution.

As a combination of connectivity and partitioning, the hybrid strategy proposed by [FL20] first connects the structurally-valid primitives, then slices the rest primitives. It can significantly reduce the complexity of the polygonal cell complex, Figure 2.10.

Chapter 3

Our Contributions

We now discuss the limitations of existing methods and present our contributions to the 3D model repairing and compact mesh reconstruction.

3.1 Repairing

Our previous reviewing in Section 2 shows that the existing repairing methods for urban models traditionally rely on local analysis and heuristic-based geometric operations on mesh data structures, which often fail to handle the diversity and complexity of errors existing in real-world models. In addition, all the mentioned repairing methods are typically tailored-made for specific 3D formats: either facet-based LOD2 CityGML models or volume-based IFC models. Therefore, our **first contribution** is proposing a general and global framework that can handle both LOD2 CityGML and IFC models while ensuring the validation of corrected models.

We observe from the defect-laden urban models that the geometric and topological errors are primarily caused by erroneous connections between adjacent surfaces or volumes. To correct these errors, one way is to rebuilt all the connection of the input facets. And this can be done by constructing a kinetic data structure [BL20] that decomposes the 3D space into polyhedral cells by extending the facets of the imperfect input model. As described in Section 4, we first convert the input polygon meshes into a soup of disconnected semantized facets, which transforms facet-based and volume-based urban models into the same form. Then the soup of facets is taken as the input of kinetic framework and generates the kinetic data structure. In the kinetic framework, the facets are extended to fill the gaps and holes, and the relations between facets are rebuilt to deal with topological errors. Once built, the polyhedral cells in the partition are regrouped by semantic classes to reconstruct the corrected output model. In addition, the semantics in IFC models are sometimes subject to interpretation, we proposed a confi-

dence score on the semantic class associated with each polyhedral cell. This gives users the option to quickly check the polyhedral cells with low confidence and interactively fix the incorrect semantic classes. We demonstrate the robustness and efficiency of our algorithm on a variety of real-world defect-laden models and show its competitiveness with respect to traditional mesh repairing techniques from both CityGML and IFC models.

3.2 Reconstruction

Our previous review demonstrates that there are two common pipelines for reconstructing compact meshes: simplifying dense mesh and assembling detected planar shapes. In an urban context, we believe that the latter is a better approach, for the following reasons:

1. In the urban scenes there are a large number of man-made objects, such as buildings and sculptures, which are almost composed of planar shapes only. Shape assembling methods can preserve the structural information that is ubiquitous in man-made objects, such as planarity and sharp features.

2. Various LODs of reconstructed urban models are required to different applications, as stated in Section 1. Shape assembling methods allow the generation of models with various LODs in an intuitive way: detecting configurations of planar shapes with different LODs by progressively reducing user-defined tolerance, Figure 2.6.

3. Some geometric regularities, such as parallelism and orthogonality, are well recognized in an urban scene. They can be preserved in the generated compact meshes by regularizing the detected planar shapes.

Among the existing methods, Kinetic Shape Reconstruction (KSR) is probably the most efficient approach that reconstructs compact meshes from point clouds by assembling detected planar shapes [BL20]. However, there exist two weak points of KSR:

1. It relies on a traditional algorithm for detecting planar shapes that delivers often bad configurations of planar shapes.

2. It requires normal orientation for extracting surfaces, which is difficult to obtain in the real world.

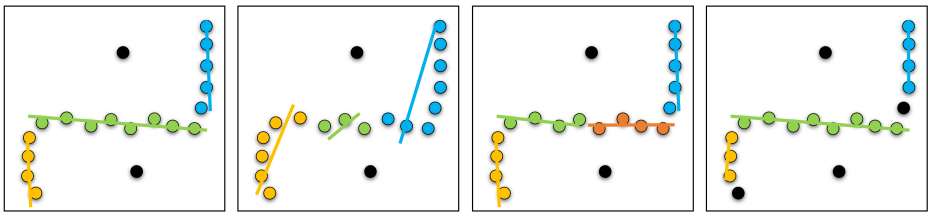


Figure 3.1: 2D Visualization of the definition of good planar shape configurations. A good configuration should have high fidelity, we prefer (a) over (b) since it has a smaller distance error between primitives and inliers. It should also be simple, (a) is better than (c) here since it contains fewer primitives. It should finally have a high ratio of inliers, (a) is more complete than (d) since it contains fewer outliers.

We contribute to compact mesh reconstruction by overcoming these two weaknesses in KSR. Determining good planar shape configurations from point clouds is our **second contribution**, as described in Section 5.1.

Planar shape detection is a long-standing problem, it consists in grouping input points into clusters while associating a parametric plane to each of them. For detecting good planar shape configurations, we first define their characteristics. The definition comes from three objectives: fidelity, simplicity and completeness, Figure 3.1. Finding a good configuration now turns out to be a trade off between these three objectives. For all the planar shape detection methods reviewed in Section 2, none of them seek for the three objectives simultaneously and none of them can operate in the whole solution space which contains a continuous plane parameters estimation space and a discrete point assignment space. We therefore proposed an algorithm that operates in the mixed discrete-and-continuous configuration space by seeking high fidelity, high simplicity and high completeness. Our algorithm consists in two key elements: (1) a simple and natural energy function and (2) an efficient exploration mechanism. The energy function measures the quality of configurations in the large solution space with respect to the three objects and the exploration mechanism uses five operators to modify a configuration under the guidance of a dynamic priority queue. In addition, our energy function and operators can be enriched. For instance, when the geometric regularity is taken into account as an additional objective, the regularity term is introduced to the energy function and the regularity oper-

ator is designed to conduct regularization, in Section 5.1.4. We demonstrate the potential of our method in a variety of scenes, from organic shapes to man-made objects, and sensors, from multiview stereo to laser, and show its efficacy with respect to existing approaches. We also demonstrate high-quality reconstructed compact meshes by using our method in place of the Region Growing method in KSR.

Facet extraction of KSR is an energy-minimizing task that requires normal orientation. We also propose a secondary contribution that lies in a practical and flexible way for extracting the facets without the use of normal orientation.

Surface extraction is solved by using graph-cut to identify each polyhedral cell in the partition with either *inside* or *outside* in KSR. However, oriented normals are needed for utilizing the graph-cut solver and they are not readily available. Obtaining occupancy solely from the coordinates of the input points is required to make the pipeline more practical. Some deep learning techniques discussed in Section 2 can estimate the occupancy of each location in 3D space taking only the point coordinates (x, y, z) as input. In Section 5.2, we employ an efficient and scalable implicit neural network (POCO) [BM22] to predict the occupancy in the bounding box of input point clouds, which is then utilized by the graph-cut solver. The compact meshes therefore can be reconstructed from only the point coordinates through combining KSR and POCO. We demonstrate the potential of the combination in a variety of scenes and objects.

Chapter 4

Repairing 3D urban models

4.1 Introduction

Computerized 3D models that recreate real urban environments play a more and more fundamental role in our everyday life for assisting us during navigation, imagining our urban projects, entertaining us with video games and movies, optimizing our telecommunication networks or the construction of our houses, reducing our energetic consumption, or protecting us by anticipating disaster scenarios. At the building scale, 3D models are usually conceptualized through the Building Information Modeling (BIM) framework, typically with the Industry Foundation Classes (IFC) format that spatially decomposes a building into volumetric objects. The boundary of these objects can be represented under the form of 2-manifold watertight polygon meshes enriched with semantic properties indicating their nature, e.g. wall, floor, door or empty space. At the city scale, Geographic Information Systems (GIS) practitioners rather rely on the CityGML formalism to represent urban objects with different Levels Of Detail (LOD), as described in [GL12]. In particular, the popular LOD2 CityGML models are typically single solids whose surface components are enriched with semantic properties. Both IFC and CityGML 3D models can thus be represented under the form of 2-manifold watertight polygon meshes whose volume or surface components are enriched with semantic properties.

During their creation (either interactively with human operators or automatically with reconstruction algorithms), these 3D models are frequently corrupted by geometric and semantic errors such as degenerated facets, self-intersections, gaps or non-manifold components that require to be corrected before use. Figure 4.1 shows such errors degrading CityGML and IFC models. Repairing these 3D models in an automated manner is an important scientific challenge. Existing methods typically detect and identify errors before correcting them by local geometric operations such as insertion, re-

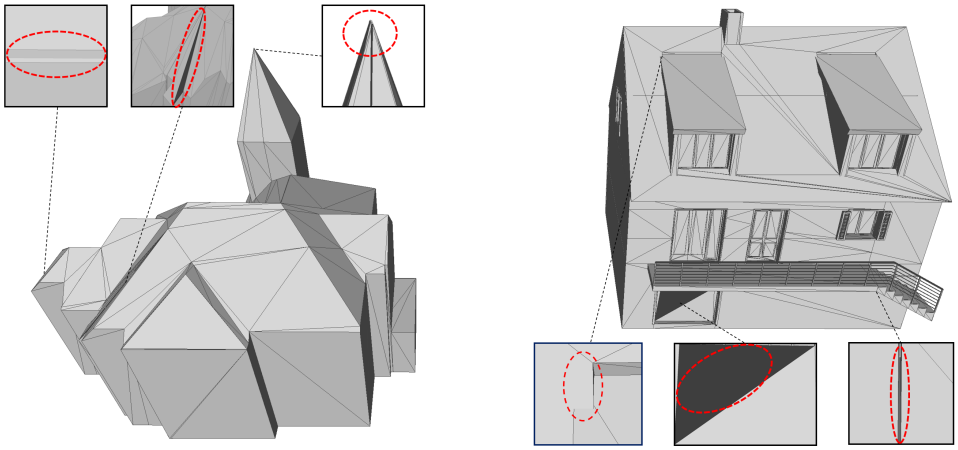


Figure 4.1: Imperfect CityGML (left) and IFC (right) urban models. The closeups show some typical geometric and semantic errors contained in the models with, from left to right: self-intersection, hole, vertex misalignment, overlapping, wrong facet orientation and gap.

moval or snapping of facets. Built on heuristics and a fine parameter tuning, these methods do not generalize well to the variety of buildings.

To address this challenge, we adopt a more global strategy based on the construction of a space partitioning data structure. Built from the facets of the defect-laden input model, the latter decomposes the 3D space into a valid embedding of polyhedra from which the corrected model is extracted. Intuitively, such a data structure can be seen as a scaffold in which all the relations between facets are re-built in a natural manner. This strategy is based on the observation that most of errors contained in 3D models originate from a wrong connectivity between adjacent facets. Such a strategy has been inspired by [DDVM14] who construct Combinatorial Maps [DL14] from soups of facets. Their construction mechanism, however, suffers from a severe drawback: based on snapping and cutting operations, it cannot guarantee to return valid combinatorial maps. In contrast, our method relies upon a recent kinetic data structure [BL20] whose construction is exact, time-efficient, parameter-free and conceptually natural: input facets extend at constant speed until colliding and forming a polyhedral partition of the 3D space. Once built, the cells of the polyhedral partition are regrouped by semantic classes, the output 3D model being defined as the set of facets at the

interface between cells of different semantic labels. This step is formulated as an energy minimization problem.

Contrary to local heuristic-based algorithms, our approach has a low number of parameters and offers a high genericity for repairing geometric and semantic errors contained in different types of urban models. We demonstrate the robustness and efficiency of our algorithm on a variety of real-world defect-laden models and show its competitiveness with respect to traditional mesh repairing techniques on both BIM and GIS data.

Our approach presents several contributions to the field. First, from a conceptual point of view, we propose a repairing system that, in contrast to conventional detection-then-correction pipelines, undo the connectivity between all facets of the input model before rebuilding them in a natural and efficient manner. Then, we present a general and flexible formulation to the semantic labeling of the polyhedral partition by seeking both consistency with the input semantics and connectivity simplicity within a Markovian energy minimization problem. Finally, we propose two application scenarios for repairing 3D models of buildings, one for surface-based models using the CityGML formalism and one for volume-based models using the IFC standard. Experiments on these two application scenarios were performed on a variety of building types such as residential, industrial, architectural and building blocks, with different input complexity ranging from 5 to $2,146$ facets for CityGML and from 978 to $142,565$ facets for IFC models.

4.2 Overview

Our algorithm requires as input a polygon mesh describing a building with a CityGML or IFC formalism. While these formalisms are different (the former is based on a surface-based representation whereas the latter relies upon a volume-based representation), one strength of our approach is to first convert these specific input polygon meshes into a general soup of disconnected semantized facets (explained in Section 4.3.1). We then make the extracted facets grow at constant speed within a kinetic framework to decompose the 3D space into a low number of convex polyhedra (Section 4.3.2). Finally, these polyhedra are regrouped according to semantic similarities within an energy minimization detailed in Section 4.3.3. The output 3D model is a polygon mesh whose facets are semantically enriched, similarly to the input

model. These three steps of our approach are illustrated in Figure 4.2.

Our algorithm is designed to correct the most frequent geometric and semantic errors contained in urban 3D models. These errors include vertex misalignment with the duplication of vertices whose coordinates are not exactly identical, inverted facets that have a wrong orientation, semantic defects with volumes associated with the wrong semantic class and general problems of self-intersections, holes or overlapping with typically a wrong connection of adjacent facets of volumes components. Note that our algorithm does not handle large missing parts that require the use of tailored-made completion algorithms.

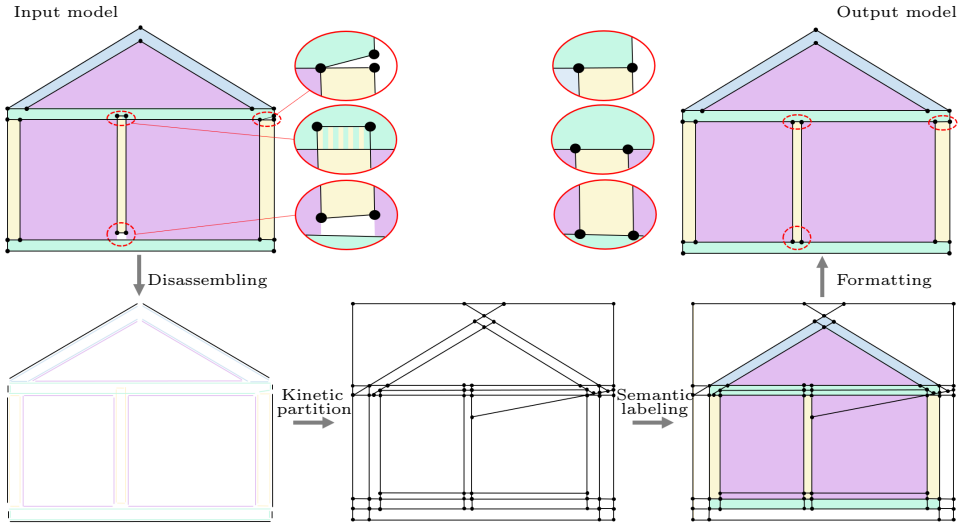
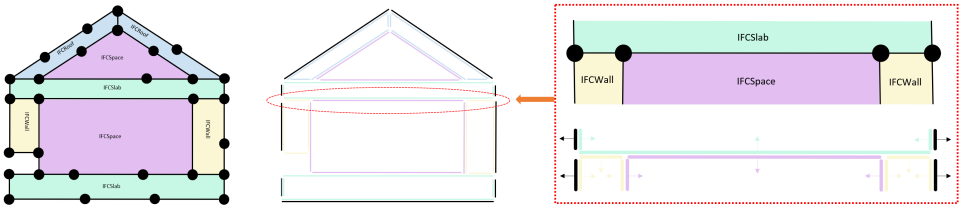


Figure 4.2: Overview of the proposed approach. Our algorithm departs from an urban 3D model corrupted by geometric and semantic errors, here a BIM model with gap, element overlapping and inaccurate vertex position (top left). We rebuild all the geometric relations between its facets by first collecting and enriching them with semantic information (bottom left) followed by constructing a kinetic data structure that partitions the space into polyhedra (bottom middle). The latter are then regrouped by using a semantic labeling formulation (bottom right) and reformatted into a BIM model in which the initial errors are corrected (top right).



(a) (b)

Figure 4.3: Disassembling of an IFC model. After collecting the boundary facets of each IfcElement (a), we assign them the semantic class of their IfcElement (b). As illustrated in the closeup, each surface component is typically associated with two facets of opposite direction and with different semantic class. Color code: space, **outside**, wall, slab, roof).

4.3 Our approach

We now describe the different steps of our algorithm and explain how it can be used on both GIS and BIM models.

4.3.1 Disassembling

The first step consists in extracting a soup of disconnected facets enriched with semantic properties from the input model. Each input facet is oriented: its normal points by construction towards the outside space in case of CityGML models and towards the center of the IfcElement to which the facet belongs in case of IFC models. Our idea is to collect each facet of the input model and to assign it the semantic class pointed by its normal.

In case of an IFC model, semantic classes are typically *roof*, *wall*, *slab*, *door*, *window*, *space* or *outside*. These classes correspond to the nature of volume elements that can be found in the IFC formalism where, for instance, the class *wall* refers to an IfcWall element. For each facet of an IfcElement, we thus simply assign it the corresponding semantic class of this IfcElement. As illustrated in Figure 4.3, IfcElements usually share common boundaries. The soup of collected facets thus contain coplanar facets with opposite orientations and assigned to different semantic classes.

In case of a CityGML model, we consider the input model as a simple boundary-based representation without real semantic classes attached to it, ie without distinction between facade and roof facets for instance. However, by assuming the output model must be watertight, we can distinguish two

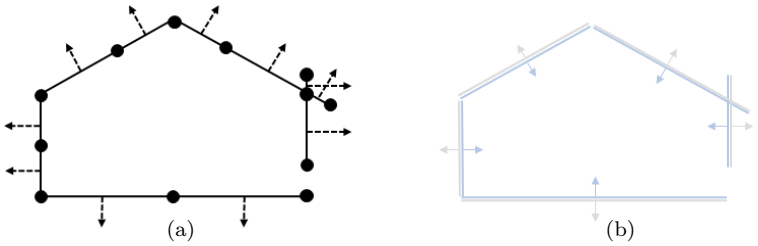
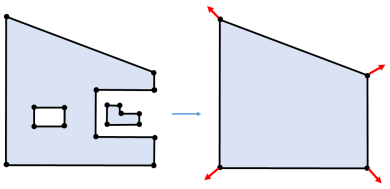


Figure 4.4: Disassembling of a CityGML model. Each facet of an input CityGML model (a) is duplicated and inverted. The class *outside* is assigned to each input facet while the class *inside* is assigned to the duplicated ones (b). Color code: input facets associated with the class *outside* in grey, duplicated ones associated with the class *inside* in blue.

types of volumes through the inside and the outside of the building. We then consider two semantic classes: *inside* and *outside* the building. As illustrated in Figure 4.4, we assign the class *outside* to each oriented facet of the input model. We also duplicate each facet, invert its normal, and assign it the class *inside*. This duplication operation is required because, contrary to IFC models that describe volume objects, CityGML models only capture the boundary surface of a building. This assignment is natural and simply relies on the assumption that the normals of CityGML input facets point toward the outside of buildings.

4.3.2 Kinetic partitioning

Our goal is now to build a kinetic data structure that partitions the space into polyhedra aligned with the collected facets.



We first regroup coplanar facets, *i.e.* facets that share the same supporting plane, and convert them into a single convex polygon. This polygon is defined as the convex hull of the coplanar facets, *i.e.* the smallest convex polygon that contains all the coplanar facets. As illustrated in the inset, input coplanar facets (left) are converted into a simpler convex polygon (right) that will be propagated in the 3D space more easily during the kinetic simulation (see red arrows). Note that converting input facets into convex polygons does not lead to a loss of information as, for each edge of an

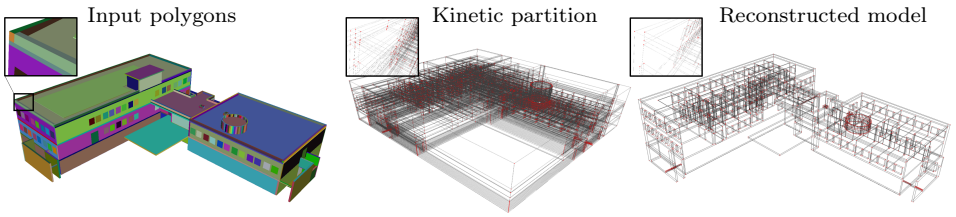


Figure 4.5: Kinetic partitioning of an IFC model. The 142,565 facets of the input defect-laden model (left) are regrouped into 668 convex polygons. The kinetic partition built from the convex polygons contains 16,306 polyhedra (middle) from which the reconstructed model is extracted (right). The IFC model contains 495 IFC elements including 209 IfcWall, 4 IfcRoof, 19 IfcSlab, 120 IfcWindow, 65 IfcDoor, 77 IfcSpace and 1 outside space.

original facet, there also exists an adjacent input facet that will be converted to a convex polygon.

We then exploit the kinetic framework proposed by [BL20] to make the convex polygons grow at constant speed until colliding with each other and decomposing the 3D space into polyhedra. The principle of such a kinetic framework has been first formalized in Computational Geometry by [Gui04]. A kinetic data structure corresponds to a set of geometric primitives, here convex polygons, whose coordinates are continuous functions of time. A kinetic framework consists in maintaining the validity of a set of geometric properties while the primitives evolve over time. When a property is no longer valid (also called an *event*), some geometric modifications are operated on the set of primitives to make the property valid again. This situation typically appears when two primitives collide. The algorithmic purpose of kinetic frameworks is to dynamically order the times of occurrence of the events within a priority queue in an efficient manner. A kinetic simulation then consists in un-stacking this queue until no primitive can move anymore. More information about this process can be found in Section 4 of [BL20].

Such a kinetic framework produces compact partitions with a low number of polyhedra. The output partition comes with strong guarantees including an exact construction with rationals, the convexity of the polyhedra and the validity of the polyhedral embedding. From an IFC model, the construction process is also highly scalable and can handle a large number of convex polygons (themselves made from potentially many input coplanar facets) in

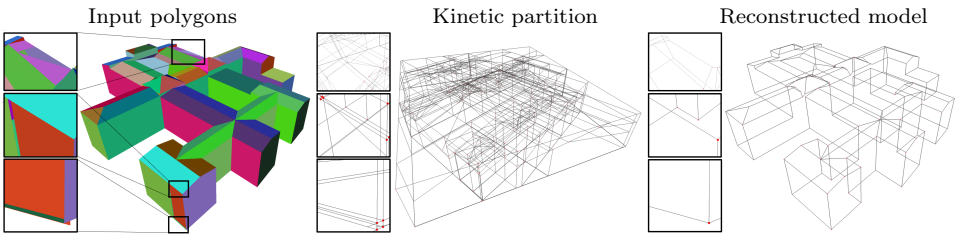


Figure 4.6: Kinetic partitioning of a CityGML model. The 378 facets (left) of the input model are grouped into 52 convex polygons, leading to the construction of a kinetic partition of 356 polyhedra (middle). The corrected model is extracted by selecting polyhedra located inside the building (right). Closeups show situations where facets are initially not well connected, the kinetic partition allowing us to properly re-build these connections.

reasonable time, Figure 4.5. Figure 4.6 shows an example of such a construction process from a CityGML model. In particular, we can see the kinetic strategy allows us to properly re-build the erroneous facet connections.

4.3.3 Semantic labeling

We now assign one of the possible semantic classes to each polyhedral cell of the kinetic partition. We formulate this problem as a multi-label energy minimization where the label set is $\mathcal{L} = \{inside, outside\}$ in case of CityGML models and $\mathcal{L} = \{slab, space, wall, roof, door, window, outside\}$ in case of IFC models. These label sets were used for our experiments, but can freely be extended to further labels. In particular, the label set for IFC models can be reduced or augmented depending on the semantic complexity of models. Such a semantic labeling is not an obvious operation as i) some facets of the kinetic partition are not present in the input model and ii) some facets of the kinetic partition can inherit from conflicting semantic information when errors are present in the input model. To solve this problem, we proposed an energy minimization framework with a multi-objective energy function.

We denote by $\mathbf{C}$ the set of polyhedral cells of the kinetic partition, and by $x_i \in \mathcal{L}$, the label that specifies the semantic class associated with polyhedral cell $i \in \mathbf{C}$. We measure the quality of a possible configuration $\mathbf{x} = (x_i)_{i \in \mathbf{C}}$ with a two-term energy of the form

$$U(\mathbf{x}) = \sum_{i \in \mathbf{C}} D_i(x_i) + \lambda \sum_{i \sim j} V_{ij}(x_i, x_j) \tag{4.1}$$

where D_i is an unary data term, and V_{ij} a potential over pairs of adjacent polyhedral cells. λ is a parameter balancing these two terms. $i \sim j$ denotes the set of pairs of adjacent cells i and j.

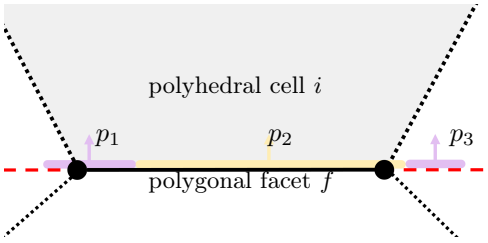
The unary data term $D_i(x_i)$ measures the consistency between the semantic class x_i assigned to the polyhedral cell i and the semantic properties of enriched facets collected during the disassembling step. More specifically, this term is formulated as a sum of local consistency measures over all the polygonal facets forming the polyhedral cell i so that

$$D_i(x_i) = \sum_{f \in i} d_f(x_i) \tag{4.2}$$

where $d_f(x_i)$ is a function measuring the local consistency on polygonal facet f. This function checks whether the enriched facets collected during the disassembling step which have the same supporting plane than polygonal facet f and oriented toward the center of mass of cell i have the same semantic class than label x_i. The function is defined by

$$d_f(x_i) = \begin{cases} 0 & \text{if } \Omega_f = \varnothing \\ \sum\limits_{p \in \Omega_f} a_f(p) \times 1_{\{x_i \neq l_p\}} & \text{otherwise} \end{cases} \tag{4.3}$$

where Ω_f is the set of enriched facets collected during the disassembling step which **i)** have the same supporting plane than the polygonal facet f, and **ii)** are oriented toward the center of mass of cell i. $a_f(p)$ is a function that measures the overlapping area between the enriched facet p and the polygonal facet f, $1_{\{.\}}$ is the characteristic function, and l_p is the semantic class associated to the enriched facet p.In the inset for instance, Ω_f is composed of three enriched facets p_1, p_2 and p_3, but only p_1 and p_2 are involved in the computation of d_f as $a_f(p_3) = 0$. The dash red line represents the supporting plane common to the three enriched facets and to the polygonal facet f (black line-segment).



The potential V_{ij} penalizes configurations that are either geometrically overly complex or semantically improbable. This term is formulated as a generalized Potts model so that:

$$V_{ij}(x_i, x_j) = a_{ij} \cdot W(x_i, x_j) \tag{4.4}$$

where a_{ij} is area of the common facet between adjacent polyhedral cells i and j and W is a matrix of size $|\mathcal{L}| \times |\mathcal{L}|$ where coefficient $W(x_i, x_j)$ indicates whether label x_i and label x_j are likely to be assigned to two adjacent cells. Three cases are distinguished to set the value of its coefficients:

- $W(x_i, x_j) = 0$ if $x_i = x_j$. This condition favors label homogeneity between adjacent cells. It allows us to encourage solutions where reconstructed volume elements are connected in a simple way.

- $W(x_i, x_j) = 1$ else if label x_i and label x_j are unlikely to be spatially next to each other, *e.g.* class *door* and class *roof* as doors are traditionally positioned vertically along walls.

- $W(x_i, x_j) = 0.1$ otherwise, *i.e.* when label x_i and label x_j are likely to be spatially next to each other while being different.

To decide whether two semantic classes are likely to be next to each other, we conducted a statistical analysis over a large range of real-world BIM models in which occurrences below 1% were considered as unlikely[1]. Figure 4.7 shows the impact of this potential term on an IFC model.

4.3.4 Formatting

The last step consists in converting the polyhedral partition in which each cell is assigned to a semantic class into a IFC or CityGML model. This operation is realized by simply grouping the adjacent cells with an identical semantic class: each resulting polyhedron gives, in case of a BIM model, an IFC element whose surface boundary is the polyhedron itself and whose type is given by the semantic class of the polyhedron. In case of a CityGML model, we simply regroup the adjacent cells labeled as *inside*. Because the kinetic partition is a valid embedding with an exact construction, the formatting step is simple and guarantees output volume elements do not overlap and share common polygonal facets. It also allows us to easily retrieve and save the adjacency relationship between the volume elements which was not originally available in the input file.

[1]For CityGML models, the notion of spatial proximity between classes is not used as only two classes are considered: we simply have $W(x_i, x_j) = 0$ if $x_i = x_j$, and $W(x_i, x_j) = 1$ otherwise.

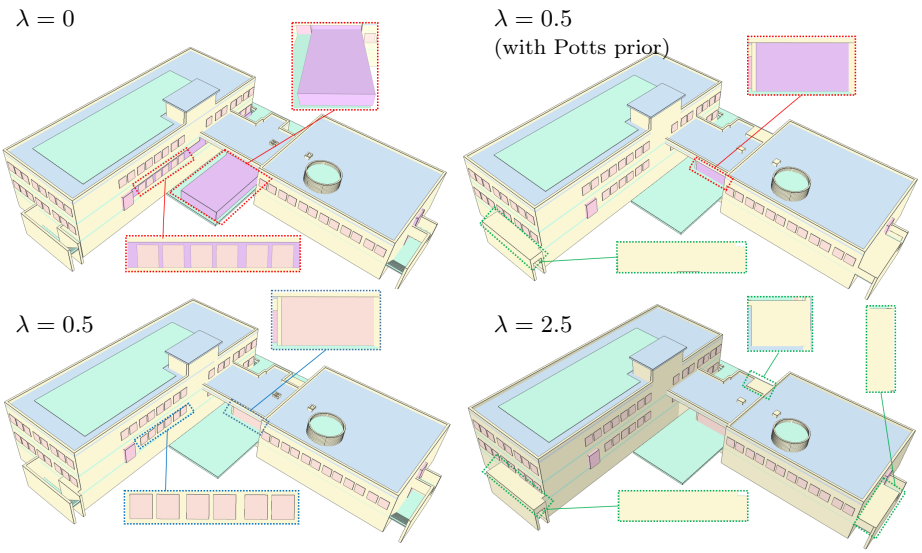


Figure 4.7: Impact of parameter λ. When only the data term is taken into account ($\lambda = 0$, top left), polyhedral cells are mislabeled at several locations, as shown on the closeups where, for instance, an outside large cell is wrongly assigned to *space*. Our potential typically allows the correction of such mislabeling ($\lambda = 0.5$, bottom left). Giving too much importance to the potential can however oversimplify the labeling and makes small elements such as windows disappear ($\lambda = 2.5$, bottom right). When considering simply a standard Potts model with a binary matrix W (top right), some unrealistic configurations can appear such as horizontal facade components.

4.4 Implementation details

The proposed approach has been implemented in C++. The *CGAL* library has been used for the manipulation of the geometric data structures and the geometric operations and the *OpenSceneGraph* library for the I/O operations on the IFC models. The efficiency of our approach relies upon several technical details that we explain below.

Coplanar facet grouping with tolerance After the disassembling step, coplanar facets are grouped and converted into a single convex polygon that will be propagated during the kinetic simulation. However, due to floating point imprecision in the coordinates of the facets, an exact coplanarity between facets is often improbable. In practice, we rather rely upon the notion

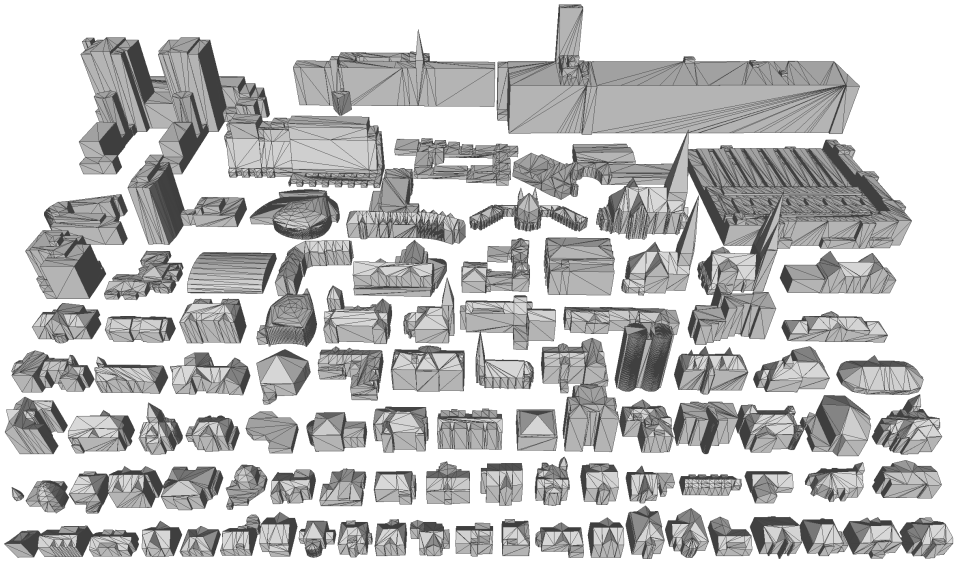


Figure 4.8: 100 LOD2 models of varying complexity collected from the Hanover 3D digital city model database.

of *near-coplanarity* by introducing two tolerance parameters. The first one defines the maximal angle between the normal vectors of two near-coplanar facets whereas the second specifies the maximal orthogonal distance between the centroids of two near-coplanar facets. Reasoning with near-coplanar facets allows us to avoid rounding issues which constitute a common source of errors in urban models [BLD+16]. In our experiments, we typically set the maximal angle to 1 degree and the maximal distance to 0.001 meter.

Simplification of some fine geometric details Input models can contain some fine geometric details, as for instance, handles on doors or windows. Such details are not important in the semantic understanding of the building and have a high geometric complexity that reduces performances of the kinetic partitioning. We thus replace these geometric details by 3D bounding boxes in our pipeline. The user then has the possibility to reintegrate these geometric details in the output models, but we do not offer specific operations to repair their potential geometric errors.

Introduction of the *outside* class for IFC models We introduce the semantic class *outside* to label the cells that are not part of the building.

This class does not exist as an IfcElement in IFC models but is necessary in our labeling formulation in order to assign a semantic label at any location in the 3D space. In practice, we duplicate all the non-overlapping facets, invert their normals and assign them the class *outside* and *space* during the disasembling step. During the labeling step, we also label the polyhedral cells located on the borders of the bounding-box with the class *outside*.

4.5 Experiments on CityGML models

We evaluated our algorithm for repairing urban models represented with the CityGML formalism, in particular with a LOD2 (Levels Of Detail) description. We used the Val3Dity tool proposed by [Led18] to test whether a model is error-free or corrupted. This tool allows us to check the validity of models against a large range of geometric and topological errors, from self-intersecting facets to vertex misalignment through non-planarity and holes.

Tests with simulated errors We first tested our algorithm from 100 buildings of varying complexity in the Hanover 3D digital City model dataset [Han21], in which we artificially introduced random errors. We selected a collection of 100 LOD2 buildings of different types, from residential houses to architectural structures through industrial estates, as illustrated in Figure 4.8. The average number of facets per building is 317. As illustrated in Figure 8, structures such as churches, industrial estates and exhibition centres are more complex and can contain up to $2,146$ facets. We then generated 300 defect-laden models from this collection, all invalid with respect to the Val3Dity tool, by randomly perturbing vertices and removing facets, leading to the generation of errors such as self-intersections, holes, or vertex misalignment. Our algorithm successfully repaired 292 of the 300 invalid models. The 8 still invalid models returned by our algorithm had typically too many missing facets to properly reconstruct the buildings, as shown in Figure 4.12. Figure 4.9 shows some buildings before and after being repaired by our algorithm. The defects such as holes, self-intersections and vertex misalignment are properly fixed, even for the buildings with a complex geometry.

Tests with real-world errors We also conducted experiments on two defect-laden CityGML datasets on the cities of Lyon, France [Met15] and

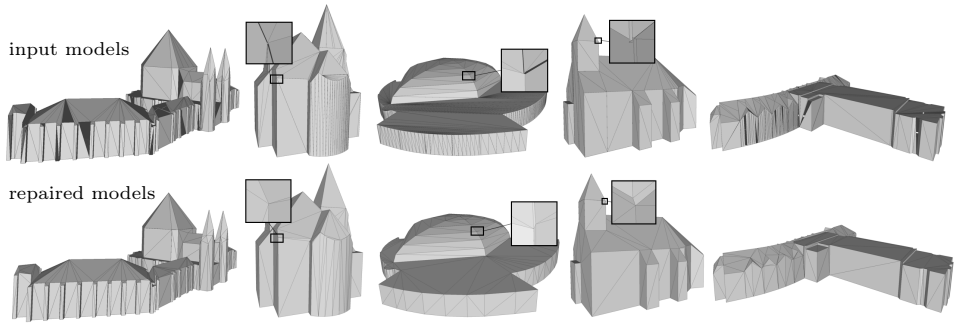


Figure 4.9: Visual results on five buildings from the Hanover 3D digital city model database. The simulated errors introduced in the input models (top) are correctly fixed by our algorithm (bottom).

Nieuw Binckhorst, The Hague, The Netherlands [The18]. They represent a dense city downtown and a complex industrial area respectively. These two datasets, illustrated in Figure 4.10, have been produced by national organizations in real-world conditions. Residential houses and warehouses are typically described by a few dozen facets while skyscrapers, churches and industrial estates can reach up to 684 facets. Most of buildings contain errors and do not pass the validity test of Val3Dity, *i.e.* 1158 out of 1421 buildings are invalid for the Lyon dataset as well as all the 414 buildings of the Nieuw Binckhorst dataset. Our algorithm achieves repairing 99.6% (respectively 92.5%) of the invalid buildings for the Lyon (respectively Nieuw Binckhorst) dataset.

Figure 4.11 shows some examples of invalid buildings with non-planar facets, vertex misalignment, holes and self-intersecting facets that have been correctly repaired by our algorithm. Besides correcting these geometric errors, our algorithm produces meshes which are guaranteed to be watertight. As illustrated in Figure 4.12, our algorithm fails to repair the input mesh when too many facets are missing or when the mesh is highly non-manifold.

Comparisons We compared our algorithm with the specialized HSW algorithm [ZLSF18] designed for repairing errors in CityGML LOD2 buildings. Three evaluation criteria were considered: (i) applicability that measures the ratio of buildings processed by the algorithm, (ii) accuracy that measures the ratio of buildings successfully corrected by the algorithm to the number of processed buildings, and (iii) the processing time. As illustrated in Table 4.1,

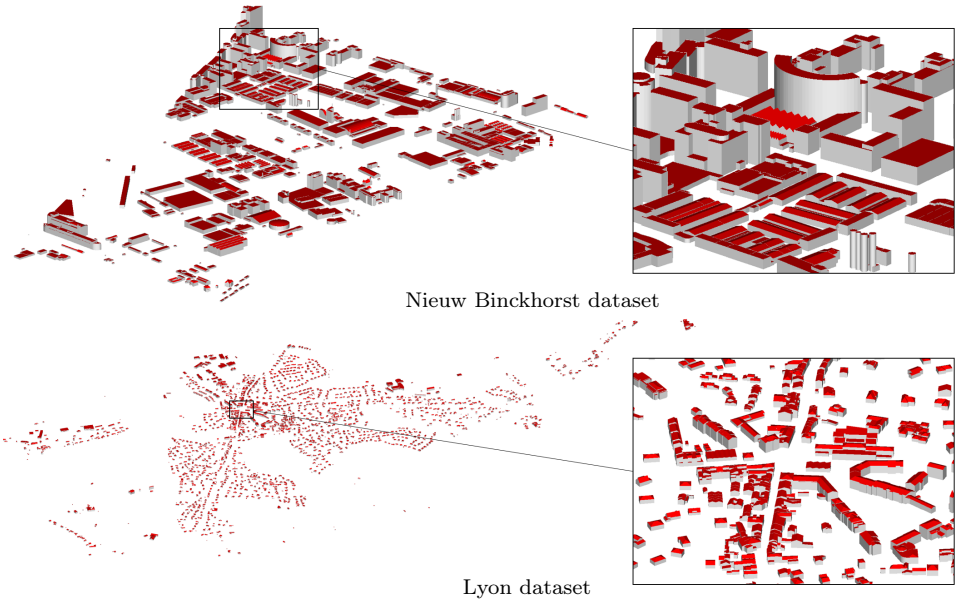


Figure 4.10: Lyon and Nieuw Binckhorst CityGML datasets. Produced by national organisms in real-world conditions, most of buildings of these two datasets are corrupted by geometric errors. 1158 out of 1421 buildings are invalid according to the Val3Dity tool for Lyon as well as all the 414 buildings for Nieuw Binckhorst. Each building in Lyon CityGML dataset is represented as *Solid* while as *MultiSurface* in Nieuw Binckhorst CityGML dataset.

our algorithm outclasses the HSW algorithm on each of the three criteria. Thanks to our global strategy that disassembles then reassembles facets, our algorithm is very general and can be applied on all the input invalid models of the Lyon and Nieuw Binckhorst datasets. This is not the case for the HSW algorithm that cannot be run on the complex and highly corrupted buildings. Our accuracy is higher by a large margin and our processing times are faster by approximately one order magnitude.

We also compared our algorithm with three generic mesh repairing techniques, *i.e.* PMP [SB20], MeshFix [Att10] and PolyMender[Ju04]. Figure 4.13 shows a visual comparison on four different CityGML LOD2 buildings. By exploiting a local repairing strategy, the surface-based methods MeshFix and PMP often generate geometric inconsistencies when input errors cannot be fixed, which often also leads to a loss of geometric details. Similarly to our approach, PolyMender is a volume-based method that offers

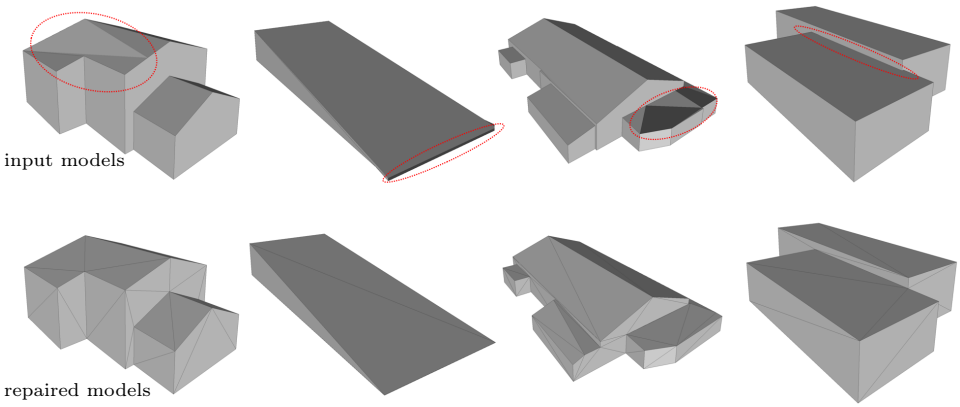


Figure 4.11: Visual results on four buildings of the Lyon and Nieuw Binckhorst CityGML dataset. Our algorithm achieves repairing geometric errors such as (from left to right) non-planar facets, vertex misalignment, holes and self-intersecting facets.

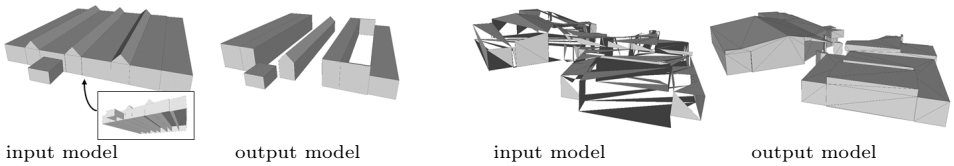


Figure 4.12: Failure cases. When the input mesh is highly non-manifold (left), some parts of the building are typically omitted. Our algorithm also fails to repair the input mesh when too many facets are missing (right).

more robustness to this problem and guarantees a watertight output mesh. However, PolyMender relies upon a volumetric grid as space-partitioning data structure that over-simplifies the output mesh and does not preserve sharp features. In contrast, our kinetic data structure is resolution-independent and not axis-aligned which allows us to repair meshes without any geometric accuracy loss compared to the input mesh. In Figure 4.13, only our method can correctly repair the four input models.

4.6 Experiments on IFC models

We tested our algorithm on IFC models. Contrary to experiments realized in Section 4.5 on CityGML models, databases of IFC models are more dif-

		Lyon	Nieuw Binckhorst
#input invalid buildings		1158	414
HSW	Applicability	50.3%	91.1%
	Accuracy	29.9%	35.8%
	Processing time (sec. per building)	2.89	10.51
ours	Applicability	100%	100%
	Accuracy	99.6%	92.5%
	Processing time (sec. per building)	0.62	0.3

Table 4.1: Quantitative comparison against the HSW algorithm [ZLSF18] on the Lyon and Nieuw Binckhorst datasets. Applicability refers to the ratio of buildings processed by the algorithm to the number of invalid input buildings. Accuracy refers to the ratio of buildings successfully corrected (i.e. valid according to the Val3Dity tool) to the number of buildings processed by the algorithm.

ficult to find. For our experiments, we rely on a set of nine IFC models of real-world buildings designed by BIM experts. This set includes five houses (three poorly detailed and two highly detailed), two tall residential structures, a single-floor office building and a highly detailed school. They have different complexity, with the number of IfcElements ranging from 10 to 509 and the number of input facets ranging from 978 to $142,565$. Because there is no tool available in the literature to test the validity of IFC models, we propose several criteria to check whether an IFC model contains some obvious geometric or semantic errors:

- Manifoldness test: We check that the boundary of each IFC element is a watertight, 2-manifold and consistently oriented polyhedron.
- Overlapping test: We check for each pair of adjacent IfcElements that their corresponding volumes do not overlap through a non-zero volume.
- Gap test: We check for each pair of adjacent IfcElements that there is no non-empty space between their corresponding volumes.
- Semantic test: We check that (i) the semantic class of each IfcElement is correct and (ii) there is no large volume inside the building not associated with an IfcElement.

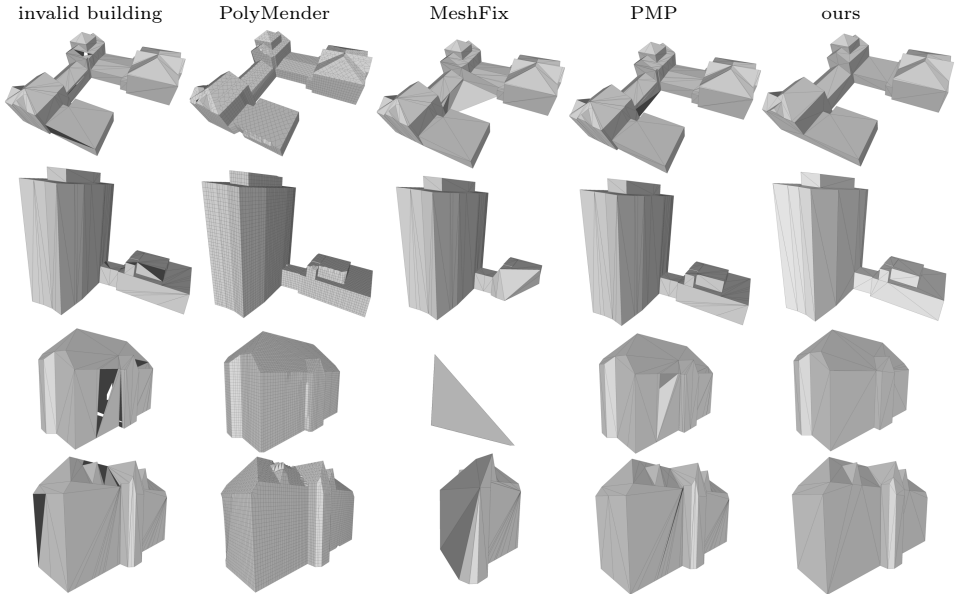


Figure 4.13: Visual comparisons with the generic mesh repairing methods PolyMender[Ju04], MeshFix [Att10] and PMP [SB20]. Only our method can correctly repair the four input models without remaining errors or loss of geometric accuracy.

We tested our algorithm on IFC models that differ in terms of complexity and levels of detail. Figure 4.14 presents visual results while Table 4.2 provides a quantitative evaluation of these results. Figure4.15 illustrates the robustness of our algorithm with the correction of challenging errors. One can see in Table 4.2 that the number of IfcElements slightly decreases after the repairing process: some input adjacent IfcElements with the same semantic class are typically merged during the semantic labeling. A strength of our algorithm is to guarantee that the boundary of each output IfcElement is, by construction, a watertight, 2-manifold and consistently oriented polyhedral surface mesh. As a result, the manifoldness test is always valid on the corrected output model. This is not the case in the input models where around half of the IfcElements typically suffers from manifoldness issues (see the #ME column in Table 4.2). Our algorithm can also fill the potential gap in between adjacent IfcElements or separate them when they overlap, as we can see in the #OE and #GE columns in Table 4.2. This is made possible thanks to the grouping of near-coplanar input facets that allows us to realign geometric approximations in a global manner. Our algorithm finally

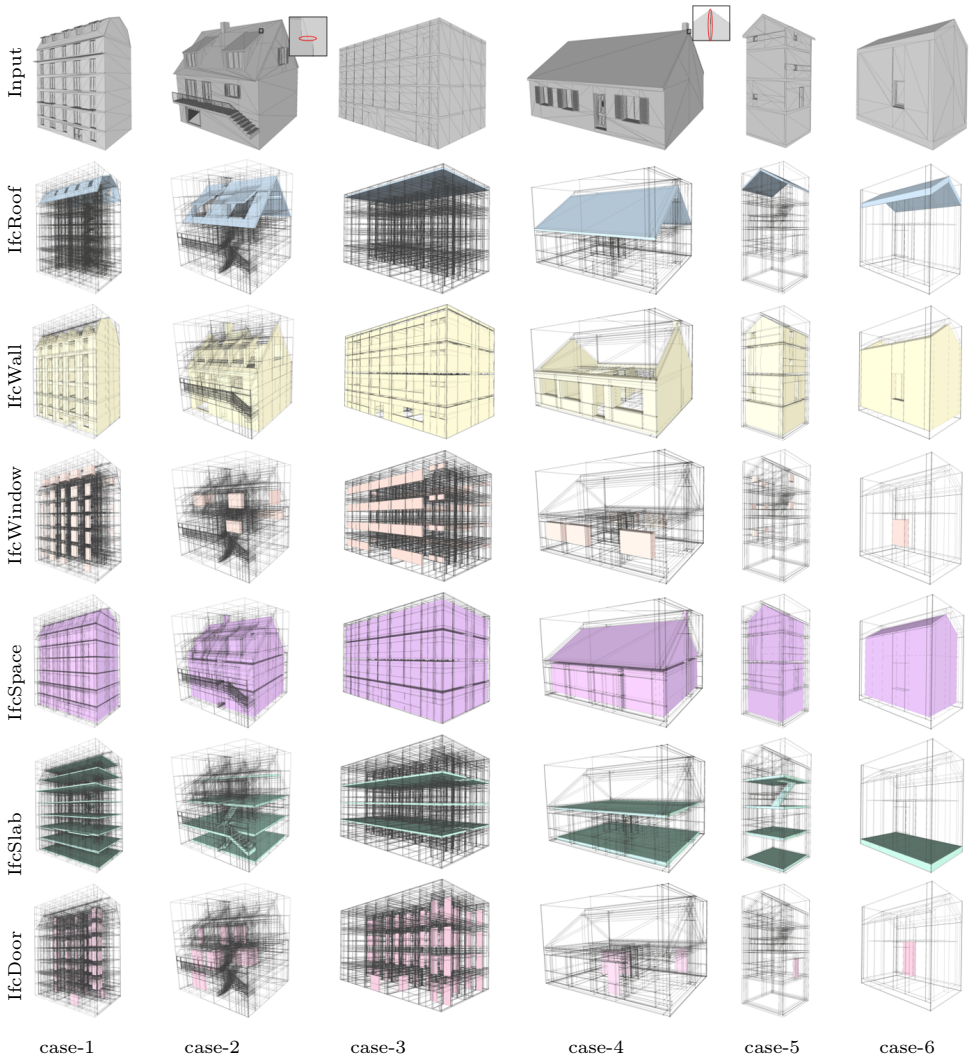


Figure 4.14: Visual results on different IFC models. Our algorithm rebuilds the various IfcElements of an input IFC model (top) while guaranteeing a valid embedding of the volumes. Input defects such as overlapping IfcElements or gaps between IfcElements (see closups) are fixed even for complex buildings. The reconstructed IfcElements of each semantic class are shown from the second to last rows inside the polyhedral partition (black wireframe). Color code: roof, wall, window, space, slab and door.

reduces the number of semantic errors, but is not able to entirely correct them. Situations where semantic errors cannot be corrected occur when the human-operator wrongly annotated a large IfcElement in the input model.

		#IfcElement	#ME	#OE	#GE	#SE
case-1	Input	509	236	12	31	0
	Output	498	0	0	0	0
case-2	Input	125	90	8	2	5
	Output	144	0	0	0	4
case-3	Input	577	182	17	1	2
	Output	554	0	0	0	0
case-4	Input	48	25	9	3	2
	Output	46	0	0	0	2
case-5	Input	51	20	3	3	1
	Output	49	0	0	0	0
case-6	Input	10	6	2	0	0
	Output	10	0	0	0	0

Table 4.2: Quantitative results on the IFC models shown in Figure 4.14. #ME (respectively #OE, #GE and #SE) refers to the number of times where the manifoldness test (respectively overlapping test, gap test and semantic test) is not valid.

	case-1	case-2	case-3	case-4	case-5	case-6
#Input facets	101967	27532	13508	12394	5012	978
#Primitives	958	598	176	120	109	28
#Polyhedra	19636	9629	13999	1188	1041	124
Kinetic partitioning (sec.)	24557.4	4507.82	108.775	24.1	18.17	0.618
Semantic labeling (sec.)	54.508	14.285	24.0995	1.437	1.3	0.232

Table 4.3: Performance of our algorithm on the IFC models presented in Figure 4.14.

Table 4.3 gives some statistics on the complexity of the IFC models presented in Figure 4.14 and the performance of our algorithm. Kinetic partitioning is typically the most time-consuming step, especially when the number of input facets is high. In order to reduce the processing time of this step, one possibility is to exploit a subdivision scheme within the kinetic framework as detailed in [BL20]. Processing time for the semantic labeling step is shorter, typically a few seconds on a medium-complexity IFC model.

Because errors in IFC models are sometimes subject to interpretation, it can be interesting to provide to the user a confidence map on the semantic class associated with each polyhedral cell. This can be done by simply choos-

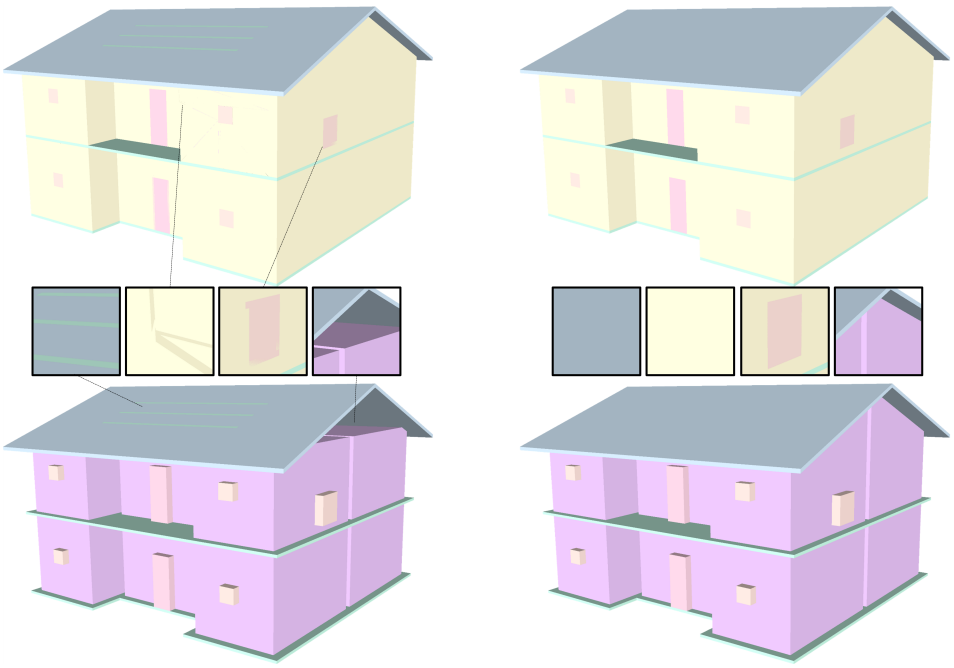


Figure 4.15: Correction of an IFC model. Our algorithm automatically corrects a defect-laden IFC model (left) containing multiple types of errors, as shown in the closeups with (from left to right) confusions between semantic classes, non-manifold situations, IfcElement overlaps and gaps. Note in particular that our corrected IFC model (right) properly fills in the large gap as IfcSpace under the roof.

ing this score as the unary data term of Equation 4.2, where lower means higher confidence. Figure 4.16 illustrates the use of such a confidence score as a way to check spatial locations where our algorithm might have taken the wrong choice. A human operator can then easily select a polyhedral cell and modify its semantic label. Such a case of user assistance is shown in Figure 4.17 with examples of semantic errors that were not corrected by our algorithm but can be easily fixed by a human operator guided by the confidence map. Note that only the results from Figure 4.17 have not been obtained automatically.

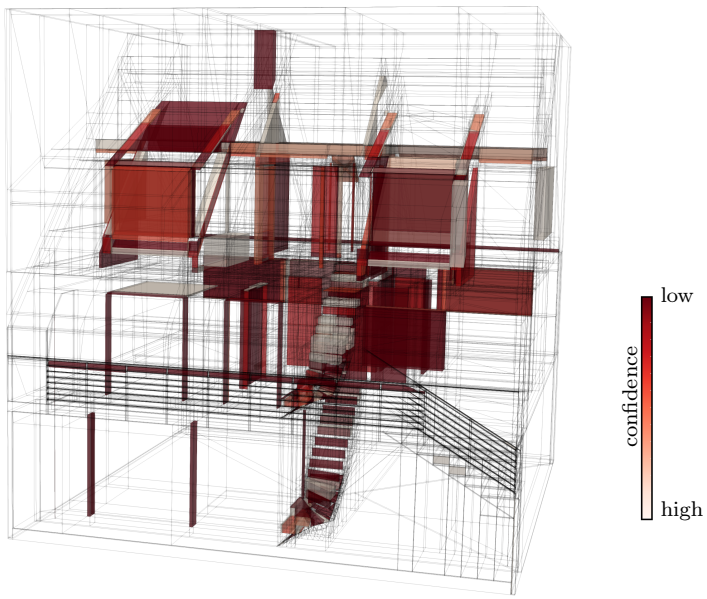


Figure 4.16: Confidence score. Our algorithm provides to the user a confidence score that highlights the locations where the semantic labeling might be erroneous. Transparent cells mean high confidence on the result whereas dark red means uncertainty.

4.7 Conclusion

We proposed an automatic approach for repairing geometric errors contained in urban models. The originality of our approach lies on the construction of a kinetic data structure that decomposes the 3D space into polyhedral cells by extending the original input facets. It allows us to re-build all the relations between the input facets in an efficient and robust manner. One strength of our approach is its capacity to adapt to different types of urban models, to CityGML and IFC models.

Our algorithm has been tested on a variety of approximately $2,100$ CityGML models with both simulated errors and real-world errors made during their conception or reconstruction. It offers a good robustness to various geometric errors including self-intersections, gaps and vertex misalignment, in contrast to specialized CityGML model repairing methods and general mesh repairing techniques. Because the latter use local geometric operators, they cannot correct non-local errors, contrary to our algorithm.

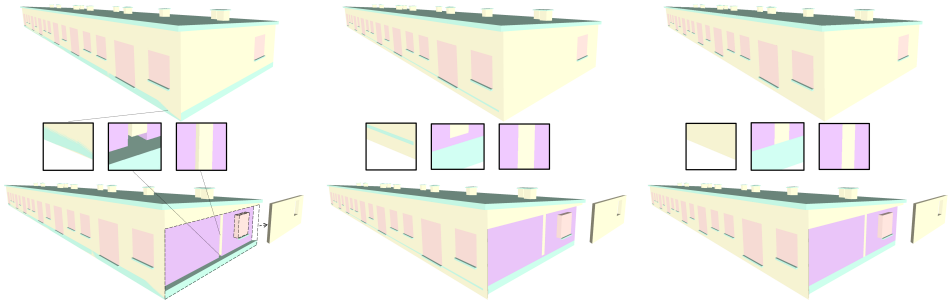


Figure 4.17: Failure cases corrected by human interactions. The input IFC model (left) contains errors (see closeups) that cannot be repaired by our algorithm (middle). These errors can be located by an human operator checking the low confidence scores and modifying the semantic classes of the concerned cells (right).

Failure cases typically occur when the input model is highly non-manifold or contains too many holes. Also, our algorithm cannot directly correct semantic errors in CityGML models as our semantic labeling formulation only makes the distinction between the inside and the outside of a building.

We also tested our algorithm on nine IFC models which have been created by BIM experts in real-world conditions. In particular, these models contain numerous geometric and semantic errors that originate from inconsistencies between adjacent IfcElements. Our experimental validation on IFC models is less extended than for CityGML model for two reasons: (i) IFC models are less common than for CityGML models and not easily accessible in practice, (ii) repairing BIM models has not been intensively studied in the literature. Our algorithm is robust to different types of errors such as gaps and overlaps between IfcElements or non-manifoldness of IfcElement boundaries. Contrary to CityGML-based experiments, our algorithm can correct some semantic errors, i.e. wrong semantic class associated with some IfcElements, thanks to the multi-class labeling formulation in which semantically improbable configurations are penalized. However, all the semantic errors cannot be repaired automatically by our algorithm. In numerous situations, the corrections of semantic errors are subject to interpretation and require the assistance of an human operator to take the good decision.

From both CityGML and IFC models, our algorithm offers good performance that originates from the use of efficient geometric data structures and

graph-based optimization techniques. In particular a complex architectural building composed of thousands facets can be repaired in a few minutes without exceeding the memory capacity of a standard laptop. A simple house with a hundred facets is processed in almost real-time.

Chapter 5

3D Compact Mesh Reconstruction

In this section, we introduce our contributions on reconstructing compact meshes, which lie in the improvements of Kinetic Shape Reconstruction (KSR) [BL20]. We first proposed a method to detect good planar primitive configurations from point clouds. The good planar primitives can lead to high-quality compact meshes. We then proposed a more practical reconstruction pipeline by combining KSR and an implicit neural network. Because of the flexibility of the pipeline, normal orientation is not necessary.

5.1 Planar primitive detection

5.1.1 Introduction

Geometric primitives are popular representations for processing massive 3D data. Such parametric shapes offer a more compact yet meaningful description than raw point clouds or dense meshes [KYZB18]. Among the most common types of primitives, planes are particularly important due to their relevance to man-made environments, e.g. urban and indoor scenes. They have been used in many data processing tasks such as data registration [ZZX+16, LL21], object recognition [RMSG21], simultaneous localization and mapping [Kae15, KCK18], structure from motion [ZJM12, RAB18], urban modeling [ZSGH18, HMFB18] and surface reconstruction [BL20, CTZ20].

Fitting planar primitives to 3D point clouds is a long-standing problem in computer vision. This is commonly formulated as a clustering operation: input points are grouped into planar components under a user-specified fitting tolerance. Solving this problem is, however, not trivial as (i) the number of primitives is unknown, and (ii) some input points, called *outliers*, are associated with no primitive as they do not fall in the fitting tolerance zones of

the primitives. Finding a *good configuration* of planar primitives is somehow arbitrary and turns out to be a tradeoff between three objectives:

1. Fidelity: the distance between a primitive and its associated input points, called *inliers*, must be small,

2. Simplicity: the number of primitives must be small,

3. Completeness: the ratio of inliers must be high.

Unfortunately, existing methods are not designed to explicitly control these three objectives simultaneously. They usually perform with one or two objectives through either incremental mechanisms [OLA16, RVDHV06, SWK07], multi-labeling energy minimization models [IB12, MMBM15, PERW16] or neural networks [LSC+21, Li,19, SLK+20] that cannot fully explore the large configuration space of this problem.

We propose an algorithm for fitting planar primitives to unorganized point clouds by seeking high fidelity, high simplicity and high completeness altogether. Our key contribution relies on the design and efficient implementation of an exploration mechanism that refines an initial configuration by minimizing a multi-objective energy function. The transitions within the large solution space are handled by five geometric operators that create, remove and modify primitives and re-assign inliers and outliers. The exploration works in tandem by alternating variational optimization at the global scale and a priority queue that sorts the local operations likely to decrease the energy. Optionally, our method offers the guarantee for not degrading the fidelity, simplicity and completeness of the initial configuration.

We show the potential of our method on different types of scenes, from organic to man-made, as well as on multiple sensors such as laser scanning and multiview stereo (MVS). We also demonstrate its efficacy with respect to existing approaches and its benefits for reconstructing compact meshes, when combined with a plane assembly method.

5.1.2 Algorithm

Our algorithm takes as input an unorganized 3D point cloud which is typically generated from MVS, laser scanning, depth cameras, or directly sampled from CAD models. The precision of the output planar primitives is

controlled by the two key parameters of traditional primitive fitting algorithms: (i) a fitting tolerance, denoted by ε, that specifies the maximal distance of an inlier to its planar primitive, and (ii) a minimal primitive size, denoted by σ, that allows primitives with a too low number of inliers to be discarded. Note that specifying these two parameters is relatively intuitive when data are defined in a meaningful system of units. It also requires less efforts than creating training sets for learning methods.

The output primitives are clusters of inlier points each associated with a supporting plane, i.e. the best least square fitting plane to its inlier points. We denote such a configuration of planar primitives by $\mathbf{x} = (\mathbf{p}, \mathbf{l})$ where $\mathbf{p}$ is a set of supporting planes parametrized in the continuous domain, and $\mathbf{l}$ is a configuration of discrete labels that indicates whether input points are outliers or inliers to one of the supporting planes of $\mathbf{p}$.

5.1.2.1 Energy formulation

We measure the quality of a primitive configuration $\mathbf{x}$ with an energy U of the form

$$U(\mathbf{x}) = \omega_f U_f(\mathbf{x}) + \omega_s U_s(\mathbf{x}) + \omega_c U_c(\mathbf{x}) \tag{5.1}$$

where terms U_f, U_s and U_c defined in the interval $[0, 1]$ account for our objectives of fidelity, simplicity and completeness respectively, and ω_f, ω_s and ω_c are positive weights balancing the terms such that $\omega_f + \omega_s + \omega_c = 1$.

Fidelity term. U_f measures the mean distance error between planar primitives and their associated inliers

$$U_f(\mathbf{x}) = \frac{1}{n_\mathbf{x}} \sum_{p \in \mathbf{p}} \sum_{i \in p} d_\varepsilon(i, p) \tag{5.2}$$

where $n_\mathbf{x}$ is the total number of inliers in the configuration $\mathbf{x}$, and $d_\varepsilon(i, p)$ represents the Euclidean distance between the inlier point i and the supporting plane p normalized by the fitting tolerance ε. Note that other metrics can be considered such as the normal deviation between the normal of the point and the orthogonal vector of the plane.

Simplicity term. U_s encourages configurations with a low number of primitives

$$U_s(\mathbf{x}) = \frac{|\mathbf{p}|}{n_\sigma} \tag{5.3}$$

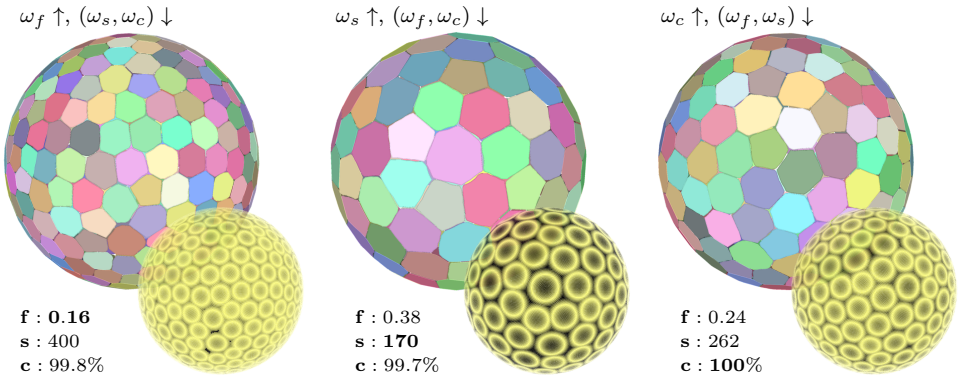


Figure 5.1: Tradeoff between energy terms. Putting weight on ω_f favors a low geometric error between planar primitives (see colored polygons respresenting their α-shapes) and input data, here 100K points uniformly sampled on a sphere (left). Increasing ω_s reduces the number of primitives (middle). A high value of ω_c produces configurations with few outliers (right). **f**, **s**, and **c** correspond to the mean Euclidean distance of inliers to their associated supporting plane normalized by ε, the number of primitives, and the percentage of inliers respectively. The colored point clouds show the geometric error distribution (yellow=0, black$\geq \varepsilon$)

where $|\mathbf{p}|$ denotes the number of primitives of the configuration $\mathbf{x}$, and n_σ is the maximal number of detectable primitives computed as the ratio of the number of input points n to the minimal number of inliers per primitive σ.

Completeness term. U_c favors configurations with a high ratio of inliers

$$U_c(\mathbf{x}) = 1 - \frac{n_\mathbf{x}}{n} \tag{5.4}$$

Energy U is formulated in a simple and natural way regarding our three initial objectives. Figure 5.1 illustrates the impact of the weights. Note that none of the three configurations can be considered as better than the others as they each perform best on one of the objectives. In our experiments, we give the same importance to each objective.

5.1.2.2 Exploration mechanism

Both continuous variables for parametrizing the supporting planes and discrete labels for grouping input points are involved in the minimization of the non-convex energy U. In order to explore efficiently such a large solution

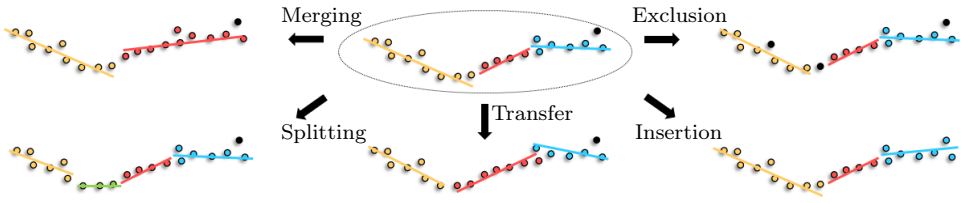


Figure 5.2: Local geometric operators. Five types of modifications can be operated from a configuration of planar primitives (top middle). The merge of two primitives (top left) and the split of a primitive (bottom left) alter the number of primitives in the configuration. The transfer of inlier points from a primitive to another (bottom middle) allows the refinement of supporting planes. Finally, the insertion of outliers as inliers of a primitive (bottom right) and the exclusion of inlier points of a primitive to outliers (top right) modify the completeness of the configuration. The inliers and supporting plane of each primitive are represented by colored points and a line-segment while outliers are displayed by black points.

space, we propose an iterative mechanism inspired by mesh decimation techniques [BKP+10]. Starting from an initial configuration $\mathbf{x}_0$, the idea consists in computing the energy variations induced by *local geometric operators* that create, remove or modify the primitives and re-assign the inliers and outliers. By sorting the energy variations of all possible operations in ascending order in a *priority queue*, we rank the operations that improve best the quality of the configuration. The algorithm then performs the geometric operation on top of the priority queue, updates the queue, and iterates until the energy does not decrease anymore.

Local geometric operators. Figure 5.2 illustrates the five types of operators used for visiting the configuration space. The operators are local and only affect one or two primitives. This condition is important to guarantee a fast computation and sorting of the energy variations. To do this, we define a notion of spatial proximity between input points and between primitives. Two points are considered as adjacent if they are connected in the k-nearest neighbor graph of the input point cloud. Two primitives are adjacent if at least a pair of their respective inlier points are adjacent.

The **transfer operator** exchanges inliers between two adjacent primitives and refines their supporting planes. This operation is performed using

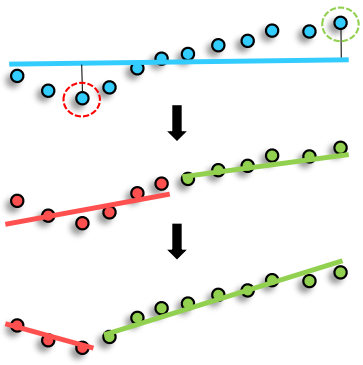
the popular K-means algorithm (with K=2) from the two inlier sets. In particular, we use the same metrics d_ε from point to plane as in Equation (5.2), and update the cluster centroids with the best least square fitting plane of the cluster of points. In order to keep primitives compact, we only transfer inliers located where the two primitives meet, i.e. the ones adjacent to inliers of the other primitive.

The **exclusion operator** reassigns the inliers far away from their supporting plane to outliers. This operation is performed by (i) sorting in descending order the distance of all the inliers to their supporting plane, and (ii) changing the k first inliers of the sorting list to outliers, k being fixed to ten in our experiments.

The **insertion operator** reassigns outlier points to the inliers of a primitive. We first list the outlier candidates whose distance to the supporting plane is smaller than the distance to the supporting plane of any other primitive while being smaller than the fitting tolerance ε. We then sort in ascending order the distance of these candidates to the supporting plane, and insert the k first outliers to the set of inliers, k being fixed to ten in our experiments.

The **merging operator** merges two adjacent primitives by reassigning their inliers to a new primitive or as outliers if located at a distance higher than ε. The supporting plane of the latter is then computed as the best least square fitting plane of its inliers.

The **splitting operator** divides a primitive into two new ones. This operator first identifies the farthest inlier on each side of the supporting plane, as illustrated by the dashed circles on the top part of the inset. The other inliers are then associated with one of these two farthest points by spatial proximity, leading to the creation of two new primitives (see red and green pairs of points and line-segment in the middle). Finally, the transfer operator is performed on the two primitives in order to refine the assignment of inliers and the supporting planes, as shown on the bottom part of the inset.



These five operators have complementary roles in the exploration. The

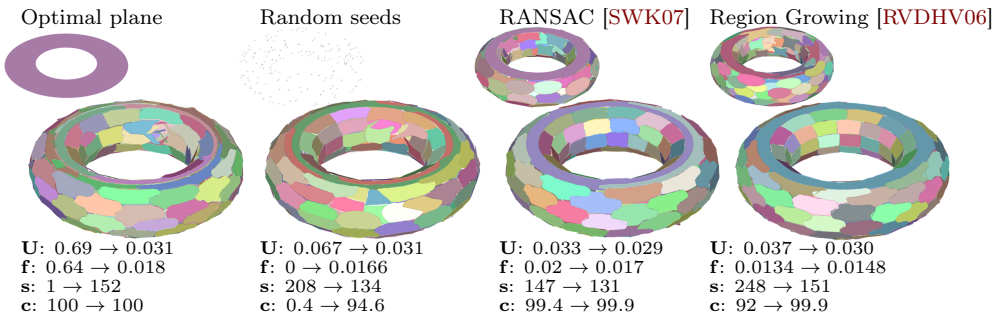


Figure 5.3: Initialization. Top row shows different initial configurations from 100K input points uniformly sampled on a torus, while bottom row shows results obtained after exploration. Starting the exploration from a good initial configuration given by incremental mechanisms such as RANSAC or Region Growing allows us to reach better configurations than from the optimal plane fitted to the input points or from many small primitives randomly distributed from input data. **U** refers to the energy of configurations.

transfer operator seeks a higher fidelity without altering simplicity and completeness. The merging and splitting operators aim at exploring configurations with different complexity whereas the exclusion and insertion operators have direct impact on completeness.

Priority queue. After each modification of the current configuration, the priority queue is updated. The concerned operation and all the operations with primitives impacted by the modification are first removed from the queue. The energy variations of all possible operations affecting the modified primitives are then computed and inserted in the queue.

Initialization. The exploration mechanism requires a good initial configuration as it finds a local minimum. As illustrated in Figure 5.3, starting from initial configurations with an already good fidelity, simplicity and completeness helps the exploration to reach better configurations. In our experiments, we use Region Growing [RVDHV06] on defect-free data and RANSAC [SWK07] on defect-laden data.

Stopping condition. The exploration mechanism stops when no more energy variation sorted in the priority queue is negative, i.e., when no operation makes the energy decrease. This condition guarantees the exploration

Algorithm 1 Pseudo-code of the exploration mechanism

```
Initialize the primitive configuration x
repeat
    Initialize the priority queue Q
    while top operation i of Q decreases energy U do
        Update x by operation i
        Update Q
    end while
    Update x by the global transfer operator
until no update modifies x any more
```

mechanism to converge quickly without bumping effects.

Details for accelerating the exploration. The transfer operator improves one objective, i.e. fidelity, without altering the other two: this particular feature makes it being called a high number of times in a priority queue. However, these local refinements between any pairs of adjacent primitives are likely undone later by the other four operators, leading to slow converge in practice. To speed up the exploration, we prefer using the transfer operator outside the priority queue in a global manner, i.e. by transferring inliers between all the pairs of adjacent primitives simultaneously. The exploration then alternates between series of priority queue updates where only splitting, merging, exclusion and insertion operations are considered, and global transfer of inliers by K-means with K fixed to be the number of primitives. Because the global transfer operator cannot degrade fidelity and modify the simplicity and completeness, the exploration cannot loop infinitely between priority queue and global transfer. Algorithm 1 describes the pseudo-code of our exploration scheme.

As illustrated in Figure 5.4, this exploration in tandem reaches similar energies as the original scheme while being one order of magnitude faster. It is also an efficient solution against a non-local simulated annealing that is three orders of magnitude slower for final configurations of identical quality.

Optional constraints. We can optionally impose that the final configuration does not degrade the fidelity, simplicity and completeness of the initial configuration. This can be done by discarding from the priority

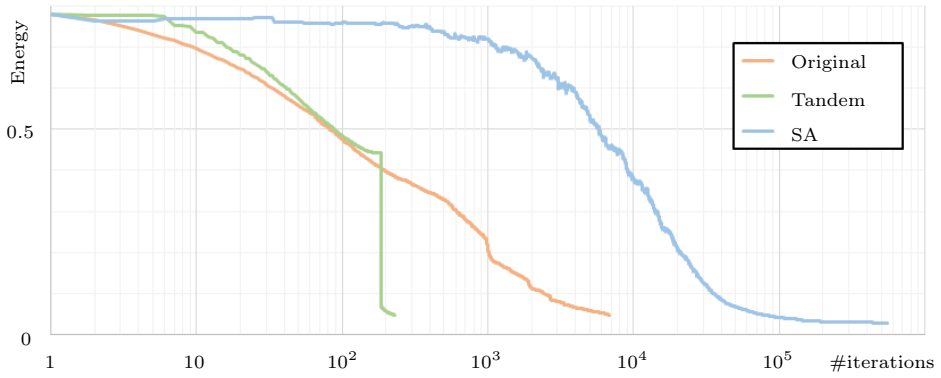


Figure 5.4: Evolution of energy U during exploration mechanisms. Our exploration in tandem (green curve) converges quickly whereas the original scheme without global transfer interruptions (orange curve) and a simulated annealing optimization (SA, blue curve) require respectively one and three orders of magnitude more iterations for reaching similar energies.

queue all the operations that lead to lower fidelity, simplicity and completeness than those of the initial configuration. In this case, the energy weights can be fixed proportionally to the initial configuration, i.e. by taking $\omega_f = K^{-1}U_f(\mathbf{x_0})^{-1}$, $\omega_s = K^{-1}U_s(\mathbf{x_0})^{-1}$ and $\omega_c = K^{-1}U_c(\mathbf{x_0})^{-1}$ where $K = U_f(\mathbf{x_0})^{-1} + U_s(\mathbf{x_0})^{-1} + U_c(\mathbf{x_0})^{-1}$. This option offers the user the guarantee not to score lower than the initial configuration on each of the three objectives, but it typically reduces the overall quality of the reached solution.

5.1.3 Experiments

Our algorithm has been implemented in C++ using the Computational Geometry Algorithms Library (CGAL). In our experiments, we typically set the fitting tolerance ε to 0.5% of the bounding box diagonal and the minimal shape size σ from 0.001% to 1% of the number of input points, depending on the complexity of scenes.

Flexibility and robustness. As shown in Figures 5.5, 5.6 and 5.9, we tested the method on various scenes and objects ranging from indoor and urban environments to statues through furniture elements. This good flexibility originates from the absence of domain-specific geometric priors in our method. We also performed tests on data generated from different types of

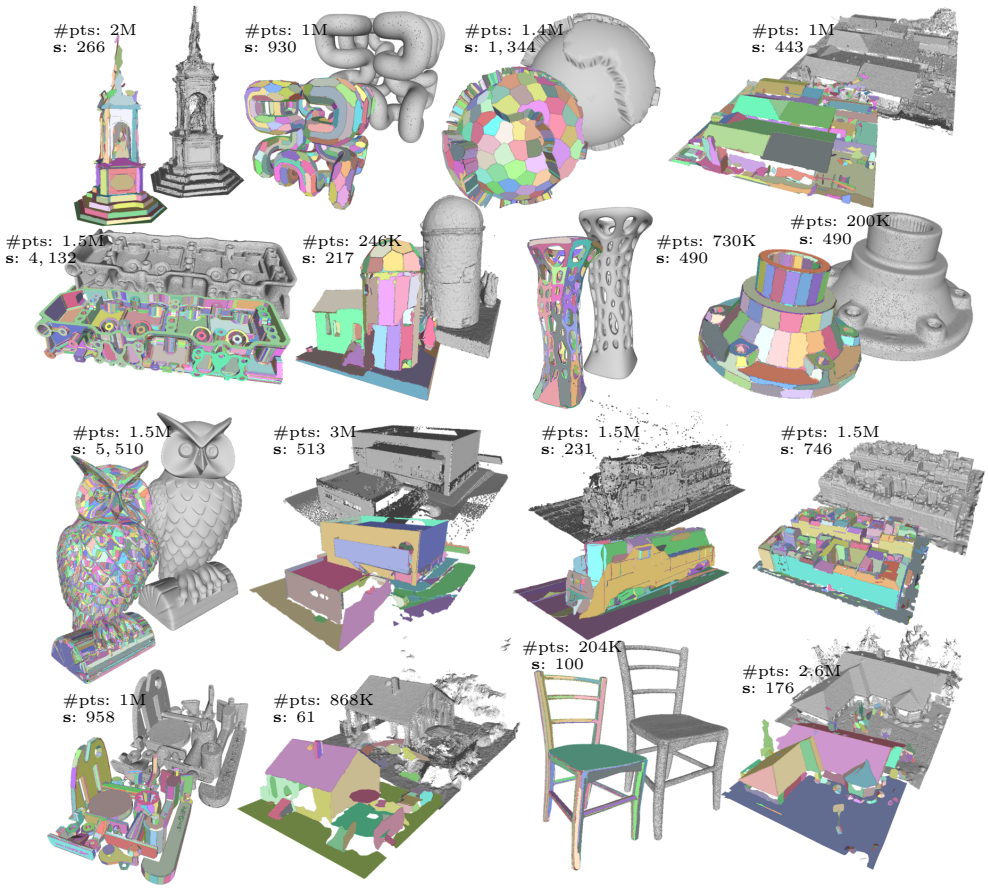


Figure 5.5: Visual results on a variety of input point clouds. From left to right, and top to bottom: Francis statue (MVS), curved Hilbert cube (CAD-based), Earth globe (CAD-based), residential House triplet (MVS), Engine (Laser), Observatory (MVS), Vase (Laser), Carter (Laser), Owl statue (Laser), Manhattan-World building (Laser), Train (MVS), Building Blocks (MVS), Mechanical system (CAD-based), basic House (MVS), Chair (Laser) and large House (MVS). #pts and **s** refer to the number of input points and the number of detected primitives respectively.

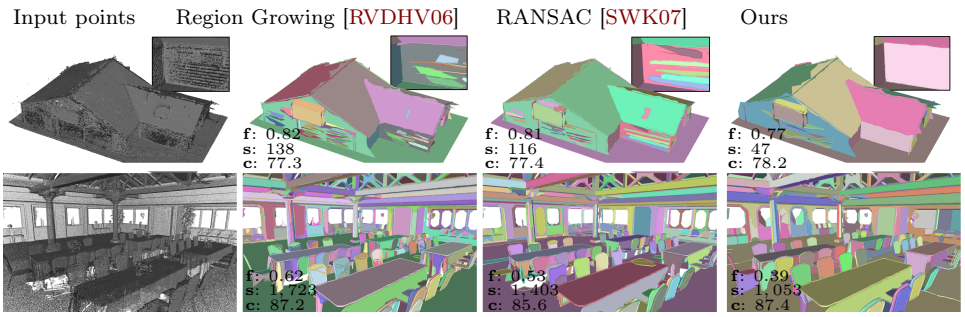


Figure 5.6: Visual comparisons with incremental mechanisms. Region Growing and RANSAC often produce inaccurate planar primitives from real-world acquisition data with, for instance, over-detected primitives on the noisy facade (top, aerial MVS) and on thin furniture elements such as chairs (bottom, indoor Laser). Our algorithm performs better by detecting fewer yet more meningful primitives without sacrificing fidelity and completeness. **f**, **s**, and **c** refer to the fidelity, simplicity and completeness scores respectively. Models from [KPZK17].

acquisition systems including Laser and MVS. Our method offers good robustness on these data, even on noisy MVS point clouds where our insertion and exclusion operators allow an efficient selection of inliers and outliers.

Comparisons. We compared our algorithm with the traditional methods Region Growing [RVDHV06], its seeding variant [OVJ+21] and RANSAC [SWK07], as well as with the deep learning methods SPFN [Li,19], ParSeNet [SLK+20] and HPNet [YYM+21]. We measure fidelity as the mean Euclidean distance from inliers to primitives normalized by the longest side of the bounding box, as proposed by [Li,19]. Simplicity and completeness are measured as the number of primitives and the ratio of inliers respectively.

We first compared our algorithm with the traditional incremental methods by using plane as the only primitive type. We tested on a sample of 5,000 models randomly chosen from the ABC dataset [KMJ+19] (each CAD model was sampled with 100K points) and on the 42 real-world models of the KSR dataset [BL20] partly based on Tanks and Temples [KPZK17]. Table 5.1 shows that our algorithm outperforms these methods on the three objectives by a large margin on the KSR dataset, and to a lesser extent, on the ABC dataset. The gain is higher from real-world data where traditional methods often produce inaccurate and overly complex configurations, as il-

		Fidelity ($\times 10^2$)	Compl.	Simpl.
KSR	RG [RVDHV06]	0.39	83.6	654.2
	SRG [OVJ+21]	0.43	83.9	612.7
	RANSAC [SWK07]	0.42	83.3	684.7
	Ours	**0.33**	**84.1**	**572.4**
ABC	RG [RVDHV06]	0.28	97.6	69
	SRG [OVJ+21]	0.30	97.1	65
	RANSAC [SWK07]	0.21	97.7	55.5
	Ours	**0.19**	**97.9**	**39.4**

Table 5.1: Comparison with traditional methods. Our algorithm achieves better scores of fidelity, simplicity and completeness on both the real-world KSR and CAD-sampled ABC datasets.

		Fidelity ($\times 10^2$)	Compl.	Simpl.
ABC*	SPFN [Li,19]	2.835	90.0	12.2
	ParSeNet [SLK+20]	0.410	99.1	8.8
	HPNet [YYM+21]	0.224	96.8	**8.3**
	Ours	**0.130**	**99.8**	**8.3**
ANSI*	SPFN [Li,19]	0.760	**95.5**	12.1
	ParSeNet [SLK+20]	1.064	91.0	**9.0**
	HPNet [YYM+21]	0.087	83.0	13.3
	Ours	**0.085**	**95.5**	9.7

Table 5.2: Comparison with deep learning methods. The ANSI* and ABC* datasets are composed of piecewise planar models only.

lustrated in Figure 5.6. In contrast, our algorithm can handle efficiently data defects, in particular by merging and splitting primitives in case of over- and under-detection, and by selecting inliers and outliers efficiently.

We also compared with the learning methods for smaller point clouds, i.e. 10K points or less. Because these methods also detect quadrics, we tested on a collection of 653 and 132 purely piecewise planar models from the ABC [KMJ+19] and ANSI [Li,19] datasets respectively. SPFN was trained on the ANSI dataset as detailed in [Li,19], whereas ParSeNet and HPNet were trained on the ABCParts dataset [SLK+20] generated from the ABC dataset. We used the end-to-end model of SPFN to predict both point assignment and plane parameters. For ParSeNet and HPNet, we combined their point assignment predictions with least square fittings for estimating plane pa-

Figure 5.7: Visual comparison with learning methods. Input points are colored per primitive cluster. Trained from another dataset, SPFN does not generalize well and exhibits many outliers (black points). ParSeNet and HPNet perform better but remain affected by frequent local mislabeling, in contrast to our method. Models from the ABC dataset.

rameters. As shown in Table 5.2, our algorithm competes well on the three metrics. The gain is particularly high on fidelity as the inlier-to-plane error is not directly controlled by a fitting tolerance parameter in learning methods. Deep learning methods also suffer from a low generalization from one dataset to the other, in particular in terms of completeness and simplicity. Only ParSeNet provides a better simplicity score than our method on the ANSI dataset, but it scores low on fidelity and completeness. Figure 5.7 shows a visual comparison on piecewise planar objects.

Choice of the fidelity metric. In our previous experiments, fidelity is measured through the traditional Euclidean distance from point to plane, but other metrics can be considered. Normal deviation can be an option when input normals are available or can be accurately estimated. Tables 5.3 and 5.4, which are an enriched version of Table 5.1 and 5.2, show quantitative results from two fidelity metrics: the traditional Euclidean distance and the normal deviation. They are used as both evaluation score and metric in the energy. As expected, fidelity score is lower when computed with the same

		Fid. L_2	Fid. normal	Compl.	Simpl.
KSR	RG [RVDHV06]	0.39	13.8	83.6	654.2
	SRG [OVJ+21]	0.43	13.8	83.9	612.7
	RANSAC [SWK07]	0.42	16.7	83.3	684.7
	Ours-L_2	**0.33**	13.3	84.1	572.4
	Ours-*normal*	0.4	**12.8**	**84.2**	**504.8**
ABC	RG [RVDHV06]	0.28	10.3	97.6	69
	SRG [OVJ+21]	0.30	10.6	97.1	65
	RANSAC [SWK07]	0.21	10.6	97.7	55.5
	Ours-L_2	**0.19**	11.8	**97.9**	**39.4**
	Ours-*normal*	0.27	**9.9**	**97.9**	41.6

Table 5.3: Enriched version of Table 5.1 with an average normal deviation score (Fid. normal, in degree) and with a variant of our algorithm using a normal deviation based metric (Ours-*normal*).

		Fid. L_2	Fid. normal	Compl.	Simpl.
ABC⋆	SPFN [Li,19]	2.835	0.869	90.0	12.2
	ParSeNet [SLK+20]	0.410	0.982	99.1	8.8
	HPNet [YYM+21]	0.224	0.988	96.8	**8.3**
	Ours-L_2	0.130	0.997	99.8	**8.3**
	Ours-*normal*	**0.026**	**0.9996**	**99.9**	8.8
ANSI⋆	SPFN [Li,19]	0.760	0.939	95.5	12.1
	ParSeNet [SLK+20]	1.064	0.85	91.0	**9.0**
	HPNet [YYM+21]	0.087	0.985	83.0	13.3
	Ours-L_2	**0.085**	0.991	95.5	9.7
	Ours-*normal*	0.097	**0.994**	**99.2**	12.1

Table 5.4: Enriched version of Table 5.2.

metric in the energy. However, the other fidelity score remains competitive with respect to the values from existing methods.

Performance. We tested our algorithm on data ranging from 8K points to 30M points. It offers a good scalability thanks to a low memory consumption. The latter results from the design of the geometric operators which are purely local. As shown in the Figure 5.8, our algorithm typically requires a few minutes for processing several millions of input points on a standard computer with a processor clocked at 2.9Ghz. Processing time of our sequential implementation of the algorithm is reasonable but remains higher

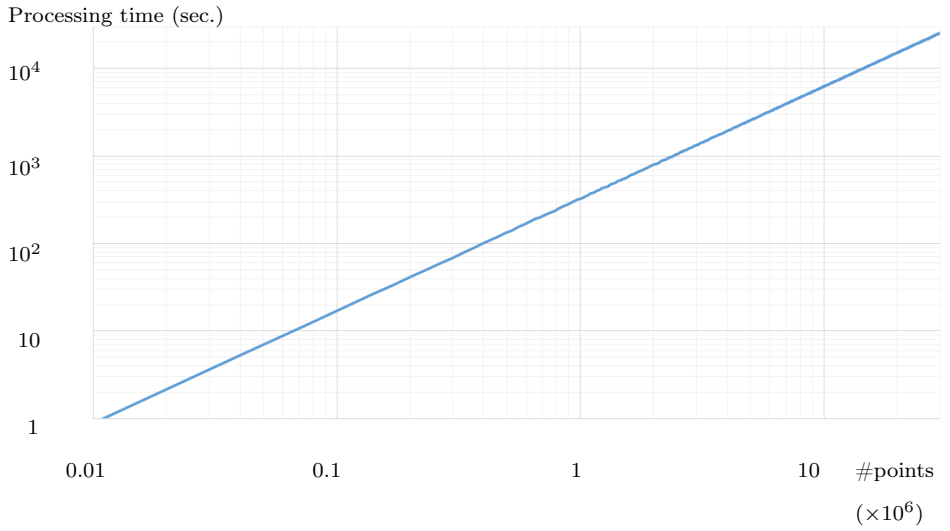


Figure 5.8: Processing time against number of points in the input point cloud.

than incremental mechanisms.

Application to compact mesh reconstruction. We applied our algorithm to the compact mesh reconstruction problem. Starting from an oriented point cloud, we used a plane assembly method [BL20] to produce a watertight, intersection-free polygonal surface mesh from the planar primitives fitted by our algorithm. Such a pipeline allows both the reconstruction of piecewise planar structures and the approximation of freeform objects.

Figure 5.10 illustrates the benefits of using our algorithm instead of Region Growing in the plane assembly method of [BL20]. Output meshes are both more accurate and more compact thanks to fewer yet more meaningful planar primitives. In particular, the shape of output polygonal facets adapts well to the local surface geometry, in coherence with the predictions of the Dupin indicatrix [Str61].

We tested the pipeline on a variety of scenes and sensors, as illustrated in Figure 5.9. It offers a good robustness to noise, outliers and, to a lesser extent, occlusions when planar components are not entirely missing in the data. The pipeline also offers a good scalability by digesting millions of points and thousands of planar primitives, as shown with the large-scale yet highly detailed results in Figure 5.9.

We compared our pipeline with the specialized compact mesh recon-

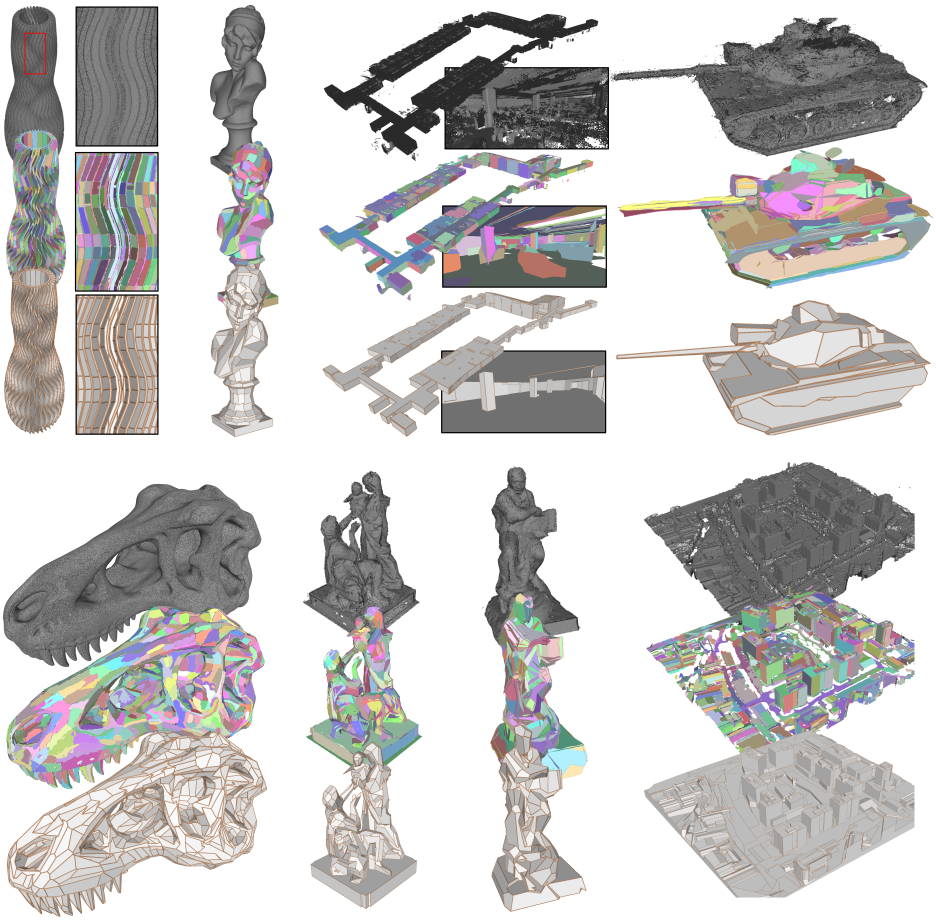


Figure 5.9: Compact mesh reconstruction and approximation on various scenes. Planar primitives detected by our algorithm (middle row) from input point clouds (top row) are assembled into compact, watertight, intersection-free polygonal surface meshes (bottom row). Such a generic and scalable pipeline produces accurate results on a variety of scenes and sensors.

struction methods PolyFit [NW17] and BSP-Net [CTZ20]. We measured the accuracy on different object categories from ShapeNet [CFG+15], the dataset used to train BSP-Net. As shown in Table 5.5, the accuracy scores of our pipeline are significantly higher on each object category. This gap comes from the low scalability of Polyfit and BSP-Net, the former being limited to the assembly of 50 planes at best and the latter imposing a volume discretization of objects by 64^3. As shown in Figure 5.11, the better scala-

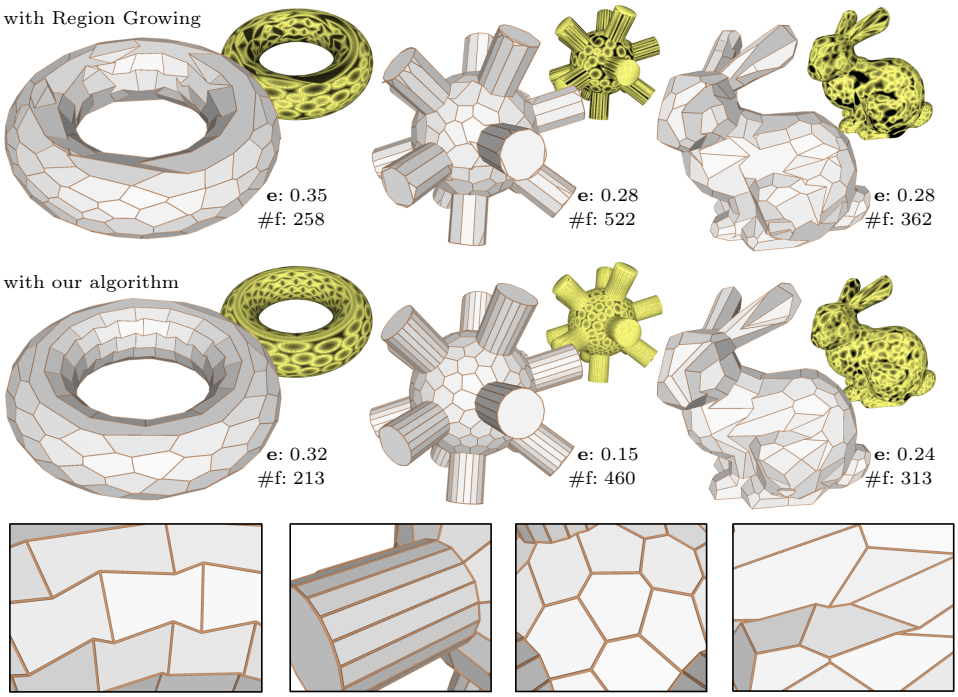


Figure 5.10: Reconstruction of freeform objects. Using our algorithm instead of Region Growing favors the assembling of planes into more accurate yet more compact polygon meshes. In particular, the shape of polygonal facets adapts adequately to the local surface geometry with concave hexagons when hyperbolic, rectangles when parabolic and convex hexagons when spherical and elliptic (closeups, from left to right). **e** and #f refer to the mean Hausdorff distance of input points to output mesh normalized by ε and the number of polygonal facets respectively. The colored point clouds show the Hausdorff distance distribution (yellow=0, black$\geq \varepsilon$)

bility of our pipeline allows the capture of fine details with highly compact meshes. Note that Polyfit and our pipeline offer the guarantee to produce watertight, intersection-free meshes, in contrast to BSP-Net that outputs a soup of convex polytopes likely to intersect.

In addition to the ShapeNet dataset, we also compared our algorithm with seven compact mesh reconstruction methods on the KSR42 dataset by following the evaluation protocol of [BL20]. In particular, we considered as evaluation metrics the symmetric mean Hausdorff distance between input points and output mesh, the mean Hausdorff distance from input points

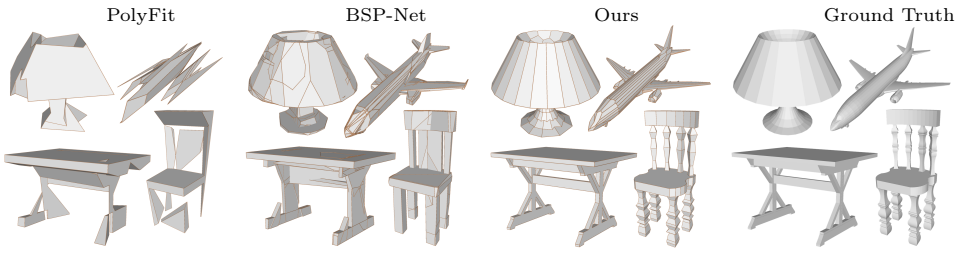


Figure 5.11: Visual comparison of compact mesh reconstruction methods. The low scalability of Polyfit [NW17] and BSP-Net [CTZ20] does not allow objects to be described with many planar components. In contrast, our pipeline can capture fine details such as the ornaments on the chair feet with highly compact meshes. Models from ShapeNet [CFG+15].

	Airplane	Car	Chair	Lamp	Table
Polyfit [NW17]	6.79	1.54	1.84	5.21	2.09
BSP-Net [CTZ20]	1.49	1.74	2.05	3.25	1.69
Ours	**0.34**	**0.38**	**0.40**	**0.34**	**0.33**

Table 5.5: Accuracy evaluation of compact mesh reconstruction methods. Accuracy is measured as the mean symmetric Hausdorff distance between input points and output meshes (values are normalized by the bounding box diagonal and multiplied by a factor of 10^2) on five object categories from the ShapeNet Dataset.

to output mesh and the mesh simplicity given by the number of polygonal facets. The seven methods include three compact mesh reconstruction methods (Polyfit [NW17], Chauve's algorithm [CLP10] and KSR [BL20]) and four surface approximation pipelines where a dense mesh is first reconstructed from input points by the Screened Poisson algorithm before being simplified either by the edge collapse method QEM [GH97], by variational shape approximation VSA [CSAD04], by plane-guided mesh decimation SAMD [SLA15] or by an alternative of the latter with corner preservation SAMD-CP. The Screened Poisson algorithm was used with an octree depth set to 9. Tables 5.6 and 5.7 provide quantitative results on respectively the simple and advanced models of the KSR42 dataset. Our algorithm outperforms the other methods on both the accuracy and the mesh simplicity in most of simple and advanced models. On a few models such as *Building C*, the KSR algorithm performs better: it typically corresponds to regular scenes where

the planar primitives detected by Region Growing have been consolidated according to parallelism and orthogonality regularities before assembling. Such geometric prior offers an advantage to KSR on such particular scenes by reducing the number of detected planar primitives. Yet, our algorithm gives better results on a large majority of models, showing that our primitive detection algorithm is more efficient than the traditional Region Growing used by the KSR algorithm. Figures 5.12, 5.13, 5.14 and 5.15 show visual comparisons on the simple models. Similarly, Figures 5.16, 5.17, 5.18 and 5.19 focus on advanced models. Note that Polyfit [NW17]and Chauve's algorithm [CLP10] do not scale well enough to produce results on advanced models.

		Barn MVS	Building A	Building B	Building C	Bunny	Chair	Cottage	Couch	Fertility-Coarse	Foam box	Hand	Laus-Coarse	Rocker arm	Rooms A	Rooms B	Temple
Type		U	U	U	U	F	S	U	S	F	S	F	U	S	I	I	S
Origin		MVS	MVS	MVS	MVS	Laser	Laser	MVS	Laser	Laser	Laser	Laser	Laser	Laser	Laser	Laser	MVS
# pts		619K	101K	73K	577K	146K	756K	143K	911K	242K	382K	369K	1.22M	733K	186K	176K	621K
ε		0.75	5	2.5	0.4	0.4	1.6	0.7	2.2	1.75	1.75	0.8	1	1.1	0.8	0.3	1.7
σ		0.08	0.6	0.25	1.2	0.15	0.5	0.2	0.7	0.2	0.2	0.07	0.8	0.04	0.3	0.6	0.3
Ours	e_A	0.151	0.658	0.587	0.593	0.321	0.529	0.308	0.458	0.284	0.222	0.340	0.454	0.190	0.526	0.543	0.296
	e_S	0.128	1.053	0.663	0.442	0.441	0.557	0.441	1.667	0.434	0.258	0.395	0.827	0.312	0.468	0.503	0.446
	#f	40	34	31	38	119	23	34	27	96	66	108	25	73	44	22	79
KSR	e_A	0.231	0.951	0.605	0.372	0.454	0.557	0.401	0.560	0.350	0.247	0.423	0.491	0.257	0.677	0.553	0.381
	e_S	0.179	1.522	0.735	0.344	0.540	0.580	0.471	1.681	0.498	0.281	0.480	0.726	0.382	0.603	0.538	0.481
	#f	38	29	23	34	111	23	28	25	95	63	92	25	73	46	26	95
Polyfit	e_A	-	1.347	2.097	0.461	-	1.447	1.491	6.385	0.388	0.221	0.460	3.977	0.361	0.590	0.680	0.548
	e_S	-	1.170	1.385	0.343	-	1.089	1.259	3.325	0.500	0.297	0.537	2.592	0.524	0.556	0.572	0.683
	#f	-	14	25	30	-	16	18	17	150	63	98	26	144	51	19	90
Chauve	e_A	1.042	9.585	2.626	0.407	0.545	1.422	1.429	3.931	0.328	0.207	0.462	0.421	0.357	0.790	0.526	0.459
	e_S	0.772	5.172	1.704	0.308	0.555	1.082	0.823	2.137	0.407	0.261	0.488	1.195	0.411	0.671	0.465	0.911
	#f	25	32	103	56	261	25	18	14	528	107	188	36	202	125	54	700
SP_QEM	e_A	2.815	4.081	6.344	2.873	1.435	4.287	2.373	2.922	0.722	2.969	0.724	2.814	2.124	2.847	2.871	2.127
	e_S	1.575	2.482	3.856	1.934	1.069	3.000	1.513	1.971	0.756	1.933	0.777	1.831	1.509	1.984	1.645	1.498
	#f	39	28	22	26	110	24	27	25	100	62	92	25	72	46	26	92
SP_VSA	e_A	0.752	4.222	2.673	1.593	1.002	1.906	1.181	1.762	1.724	1.404	1.058	2.894	1.201	2.066	0.911	0.894
	e_S	0.538	3.107	1.921	1.316	1.031	1.664	0.958	1.806	1.287	0.960	0.918	2.355	1.091	1.422	0.704	0.806
	#f	717	213	157	947	175	420	528	113	93	682	127	31	83	568	123	270
SP_SAMD-CP	e_A	1.164	1.138	0.806	0.387	1.042	1.797	1.341	1.965	0.366	2.363	0.745	0.842	0.907	0.704	0.760	1.361
	e_S	0.675	0.988	0.781	0.307	0.909	1.302	0.833	1.556	0.383	1.480	0.789	0.761	0.744	0.604	0.552	0.998
	#f	141	95	92	267	162	40	75	35	352	105	120	102	137	125	98	125
SP_SAMD	e_A	1.428	1.524	1.071	1.885	0.585	2.326	0.801	1.483	0.657	1.546	0.508	1.686	0.800	1.240	1.907	0.686
	e_S	0.902	1.059	1.240	1.359	0.820	1.394	0.776	1.840	0.907	1.481	0.806	1.721	0.971	0.940	1.186	0.766
	#f	39	28	23	32	112	21	29	26	96	64	92	25	72	46	26	93

Table 5.6: Quantitative comparisons on the simple models of the KSR42 dataset. Types of objects include urban (U), freeform (F), indoor (I) and structured (S). Origins of point cloud include multiview stereo (MVS), laser scanning (Laser) and point sampled from CAD models (CAD). The size of input points #pts ranges from 101K to 1.2M points. The fitting tolerance ε and the minimal primitive size σ are expressed in percent of the bounding box diagonal and in percent of the total number of input points respectively. The evaluation metrics e_S, e_A, #f refer to the symmetric mean Hausdorff error (in % of the bounding box diagonal), the mean Hausdorff error from input points to output model and the number of output facets respectively. Note that the scalability of Polyfit is too low to return a result on Barn MVS and Bunny after several hours of computing.

		Asian dragon	Castle	Church	Courthouse	Euler	Fertility-Fine	Full thing	Meeting Room	Navis	Tower of Pi
Type		F	U	I	U	I	F	S	I	I	F
Origin		Laser	CAD	Laser	Laser	Laser	Laser	CAD	Laser	Laser	CAD
# pts		3.6M	737K	31.1M	1.9M	2.7M	242K	1.4M	3.1M	3.6M	2.9M
ε		0.25	0.1	0.8	0.8	0.1	0.2	0.2	0.1	0.1	0.1
σ		0.004	0.001	0.005	0.002	0.004	0.03	0.004	0.003	0.005	0.001
Ours	e_A	0.057	0.002	0.372	0.088	0.057	0.046	0.060	0.197	0.055	0.013
	e_S	0.064	0.033	0.239	0.136	0.094	0.086	0.062	0.146	0.048	0.031
	#f	3210	748	387	2157	1118	831	1788	1449	546	12749
KSR	e_A	0.065	0.006	0.460	0.134	0.069	0.061	0.064	0.198	0.061	0.020
	e_S	0.069	0.038	0.292	0.183	0.108	0.095	0.064	0.146	0.057	0.039
	#f	3132	711	394	1795	1317	998	1807	1491	457	12059
SP_QEM	e_A	0.071	0.044	0.292	0.155	0.148	0.082	0.170	0.299	0.167	0.135
	e_S	0.072	0.082	0.345	0.245	0.237	0.117	0.141	0.204	0.146	0.098
	#f	3132	712	394	1795	1318	1000	1806	1490	458	12050
SP_VSA	e_A	0.125	0.098	0.569	0.254	0.242	0.183	0.378	0.202	0.192	0.295
	e_S	0.108	0.110	0.377	0.262	0.240	0.243	0.245	0.160	0.164	0.241
	#f	3351	1894	494	1818	4070	982	4368	4949	522	27285
SP_SAMD-CP	e_A	0.114	0.065	0.187	0.154	0.169	0.150	0.117	0.330	0.098	0.104
	e_S	0.101	0.078	0.214	0.195	0.166	0.152	0.094	0.215	0.079	0.110
	#f	3672	1600	1924	1835	1450	1709	10046	1552	1898	30654
SP_SAMD	e_A	0.088	0.073	0.238	0.113	0.128	0.128	0.371	0.185	0.113	0.194
	e_S	0.094	0.095	0.274	0.223	0.199	0.163	0.258	0.168	0.114	0.205
	#f	3132	733	396	1800	1318	999	1809	1491	457	12060

Table 5.7: Quantitative results on advanced models. Note that Polyfit [NW17]and Chauve's algorithm [CLP10] do not scale well enough to produce results on advanced models.

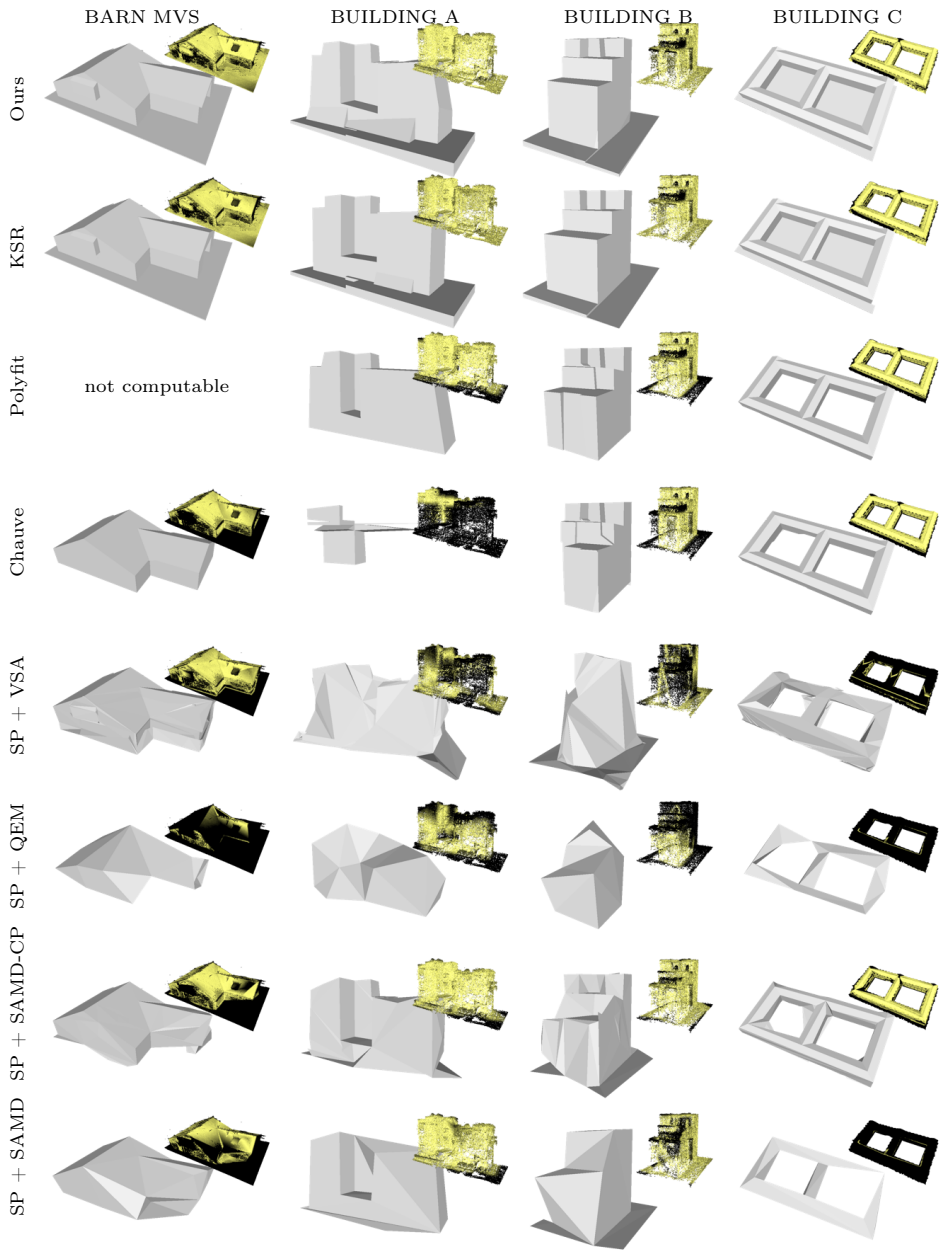


Figure 5.12: Visual comparisons on simple models (part 1/4). The colored point clouds correspond to the distribution of the Hausdorff distance from input points to output models (yellow=0, black$\geq \varepsilon$). Evaluation scores associated with these results are given in Table 5.6.

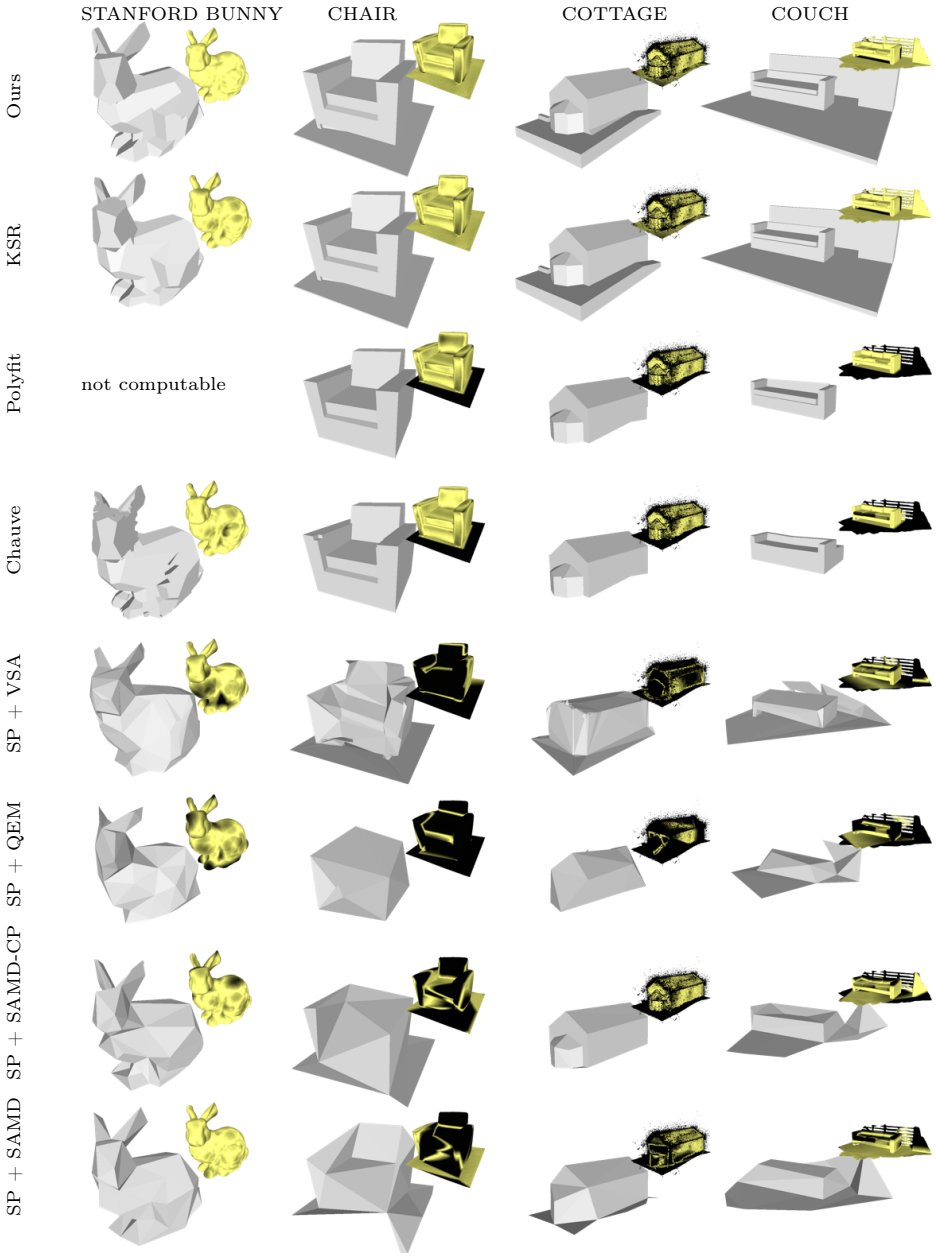


Figure 5.13: Visual comparisons on simple models (part 2/4).

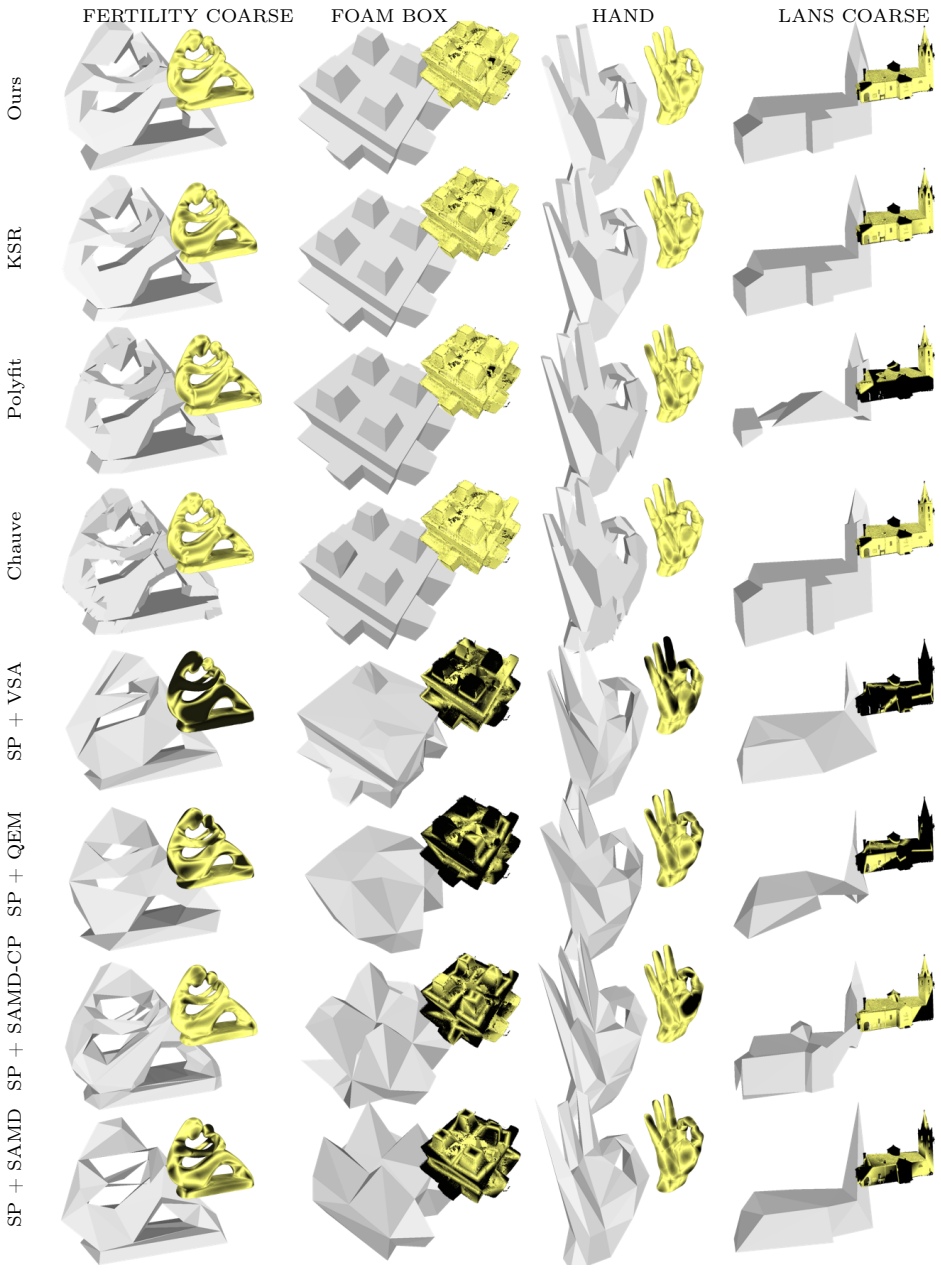


Figure 5.14: Visual comparisons on simple models (part 3/4).

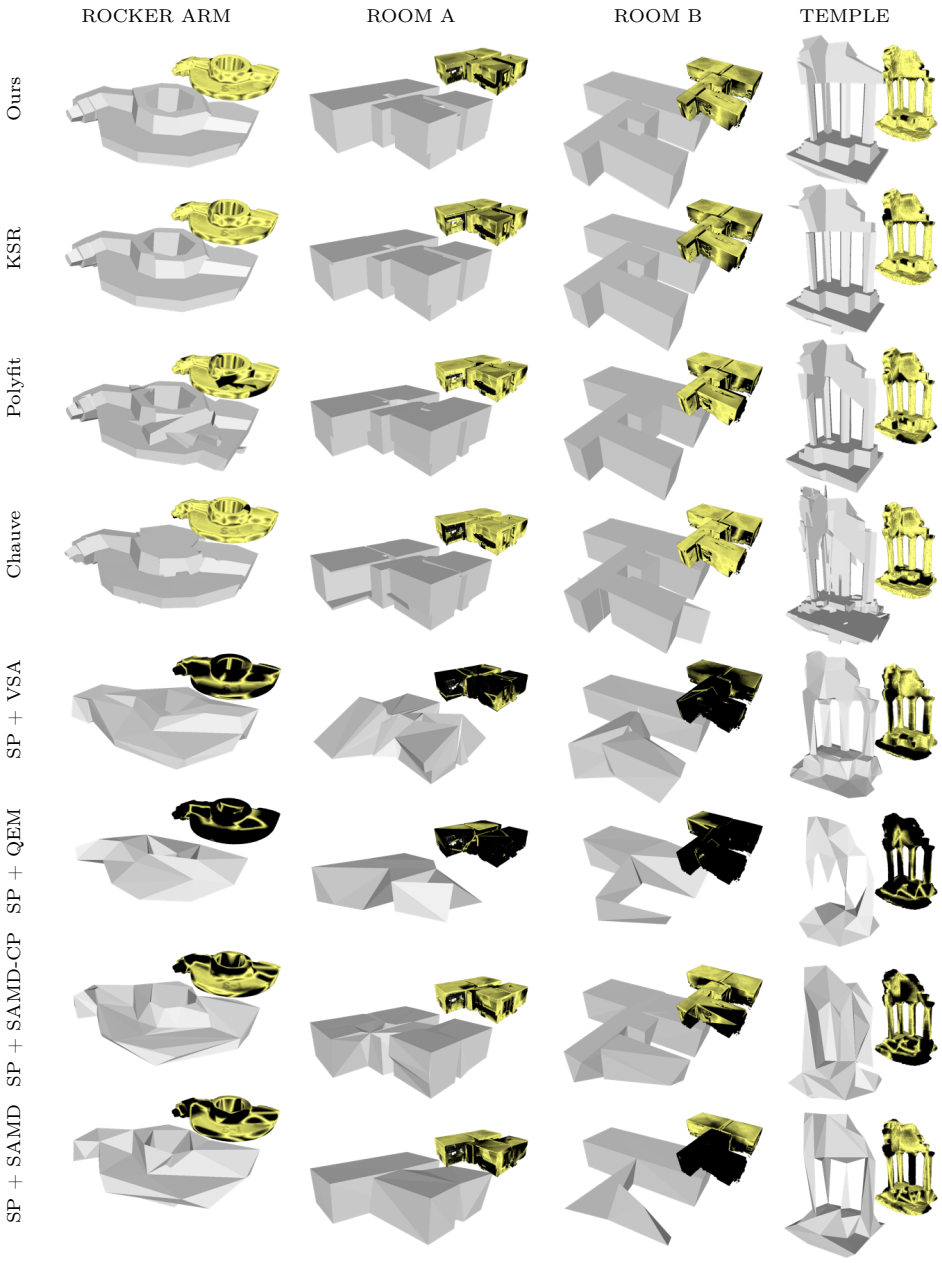


Figure 5.15: Visual comparisons on simple models (part 4/4).

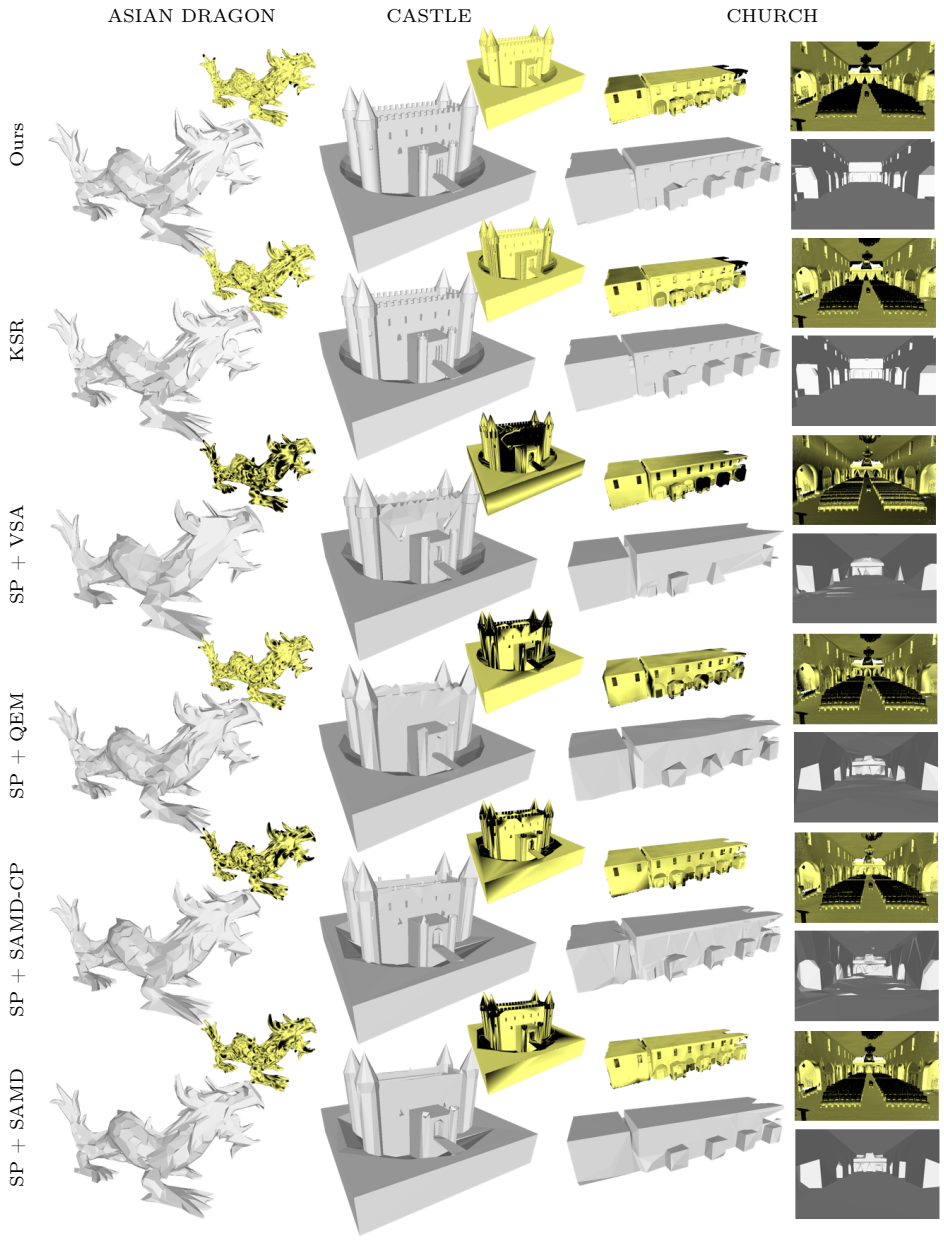


Figure 5.16: Visual comparisons on advanced models (part 1/4). Evaluation scores associated with these results are given in Table 5.7.

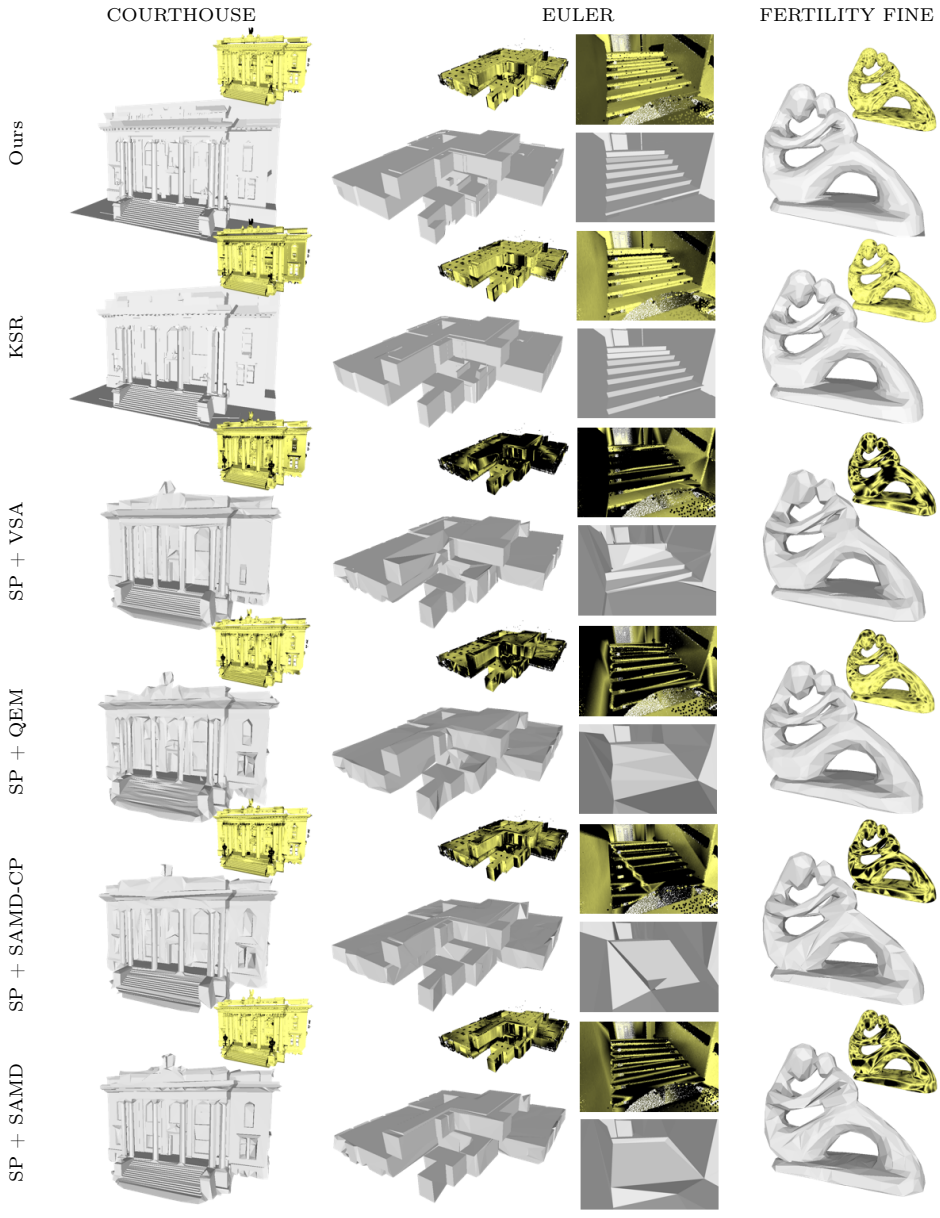


Figure 5.17: Visual comparisons comparisons on advanced models (part 2/4).

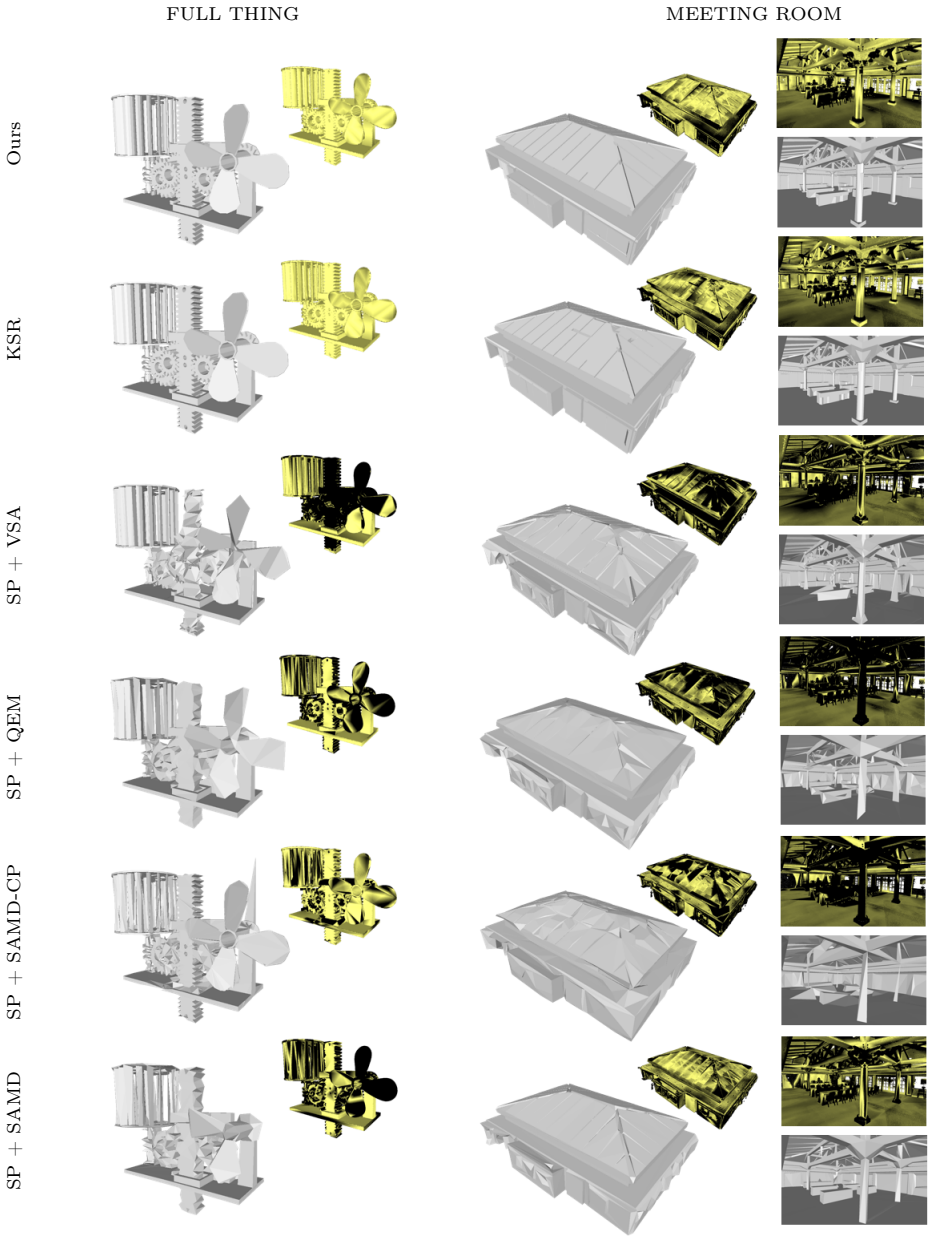


Figure 5.18: Visual comparisons on advanced models (part 3/4).

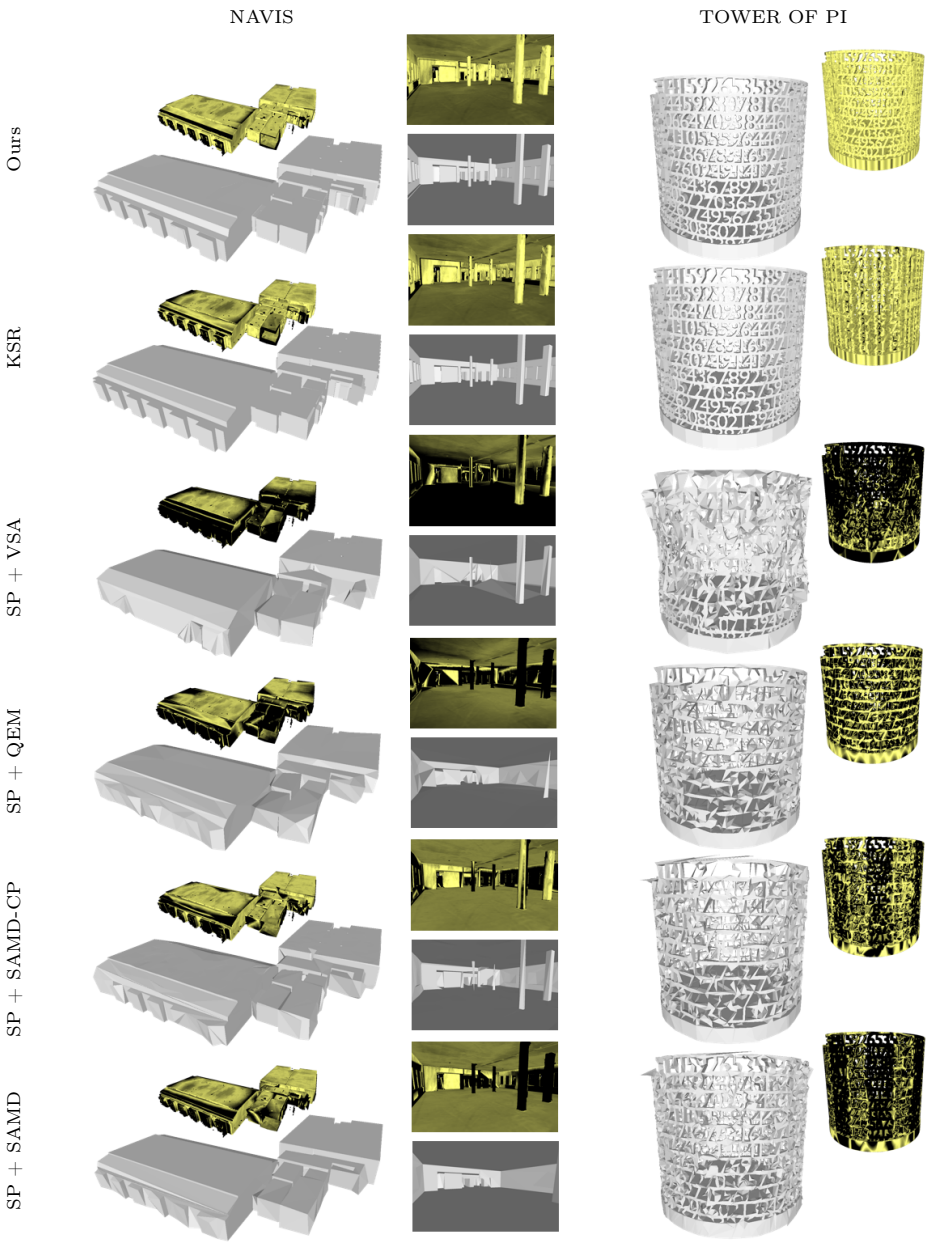


Figure 5.19: Visual comparisons on advanced models (part 4/4).

Reconstruction at different levels of details. As mentioned in Section 1, multiple LODs of the reconstructed compact mesh are required. Our method can optimize the planar shape configurations under any user-defined parameters, which can lead to high-quality compact meshes with different LODs. Figure 5.20 shows an example of a freeform object reconstruction at different LODs obtained by modifying the fitting parameters (ε, σ). Increasing progressively both the fitting tolerance ε and the minimal primitive size σ gives coarser configurations of planar primitives, and after assembling, more concise polygon meshes.

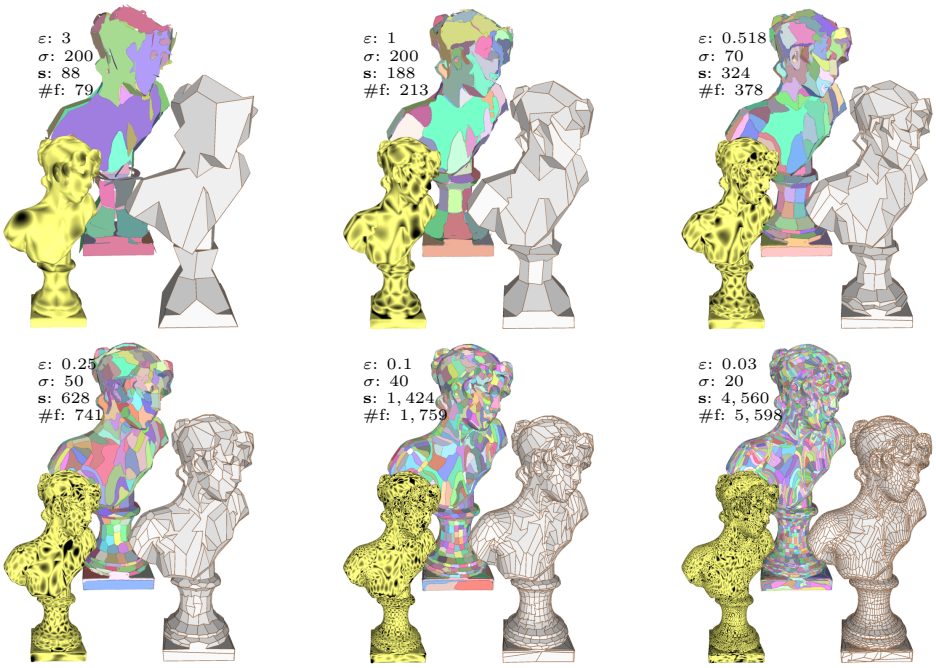


Figure 5.20: Reconstruction of a freeform object at different levels of details. By progressively decreasing the fitting parameters (ε, σ), our algorithm outputs a series of more and more accurate and complex planar primitive configurations (colored polygons, from left to right, and top to bottom). When combined with a plane assembly method, it produces polygon meshes at different levels details. **s** and #f refer to the number of primitives and the number of polygonal facets of the reconstructed mesh respectively. The colored point clouds show the distribution of the Hausdorff distance of input points to output mesh normalized by ε (yellow=0, black$\geq \varepsilon$).

Limitations. Because the exploration mechanism is local and energy U is not convex, the quality of the initial configuration influences results. Starting from Region Growing or RANSAC is a fast and scalable solution, but it might not be an optimal choice on data highly corrupted by occlusions. That said, our algorithm will also benefit from future advances in the field to produce even better results. Processing time on massive point clouds also remains high in comparison with the traditional incremental mechanisms. This can be penalizing when planar primitives must be quickly detected as a preprocessing step for a 3D vision problem. Finally, our algorithm is not designed to preserve geometric regularities. For example, the top and bottom sides of the torus in Figure 5.10 do not have exactly symmetric layouts of planar primitives. Taking into account such knowledge in our framework could be done by considering the geometric regularity as a new objective, Section 5.1.4. However, this would require a preliminary detection of regularities, which is not a trivial task in practice.

5.1.4 Extension with geometric regularization

In this section, we demonstrate the extensibility of our method by introducing geometric regularity as a fourth objective. As previously indicated, our algorithm is unable to preserve geometric regularities when only fidelity, simplicity and completeness are considered as objectives. However, it is flexible enough to introduce other objectives, as long as the new energy terms can be formulated to quantify them and the associated geometric operators can be designed to explore the solution space. A regularity term and a regularity operator are proposed.

5.1.4.1 Algorithm

Our algorithm preserves the geometric regularities by using a regularity operator to adjust the nearly regular primitives to be exactly regular. In addition, the regularity operator is guided by an updated energy function which introduces degrees of freedom into the energy function 5.1. Here, we consider three types of regular relationships: parallelism, orthogonality and co-planarity.

Regularity operator. To preserve the geometric regularities in the objects or scenes, the regularity operator should be proposed to carry out the regularization of the primitives. Instead of operating on a single primitive or

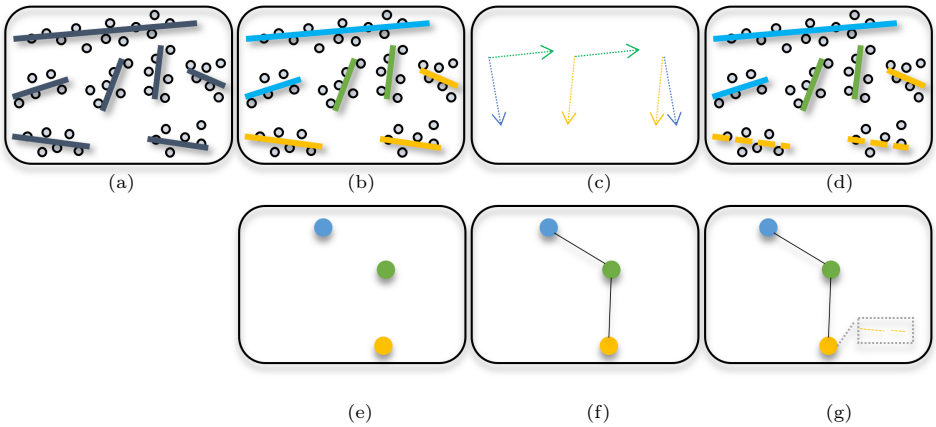


Figure 5.21: Regularity graph creation. From a planar primitive configuration (a) where all the support planes of the primitives are the best least square fitting plane to their inlier points, three *parallel cluster*s are generated (b) as nodes in the regularity graph (e). Then the orthogonal relationships are detected by iteratively checking the mean normals of a pair of *parallel cluster*s (c), the nearly orthogonal *parallel cluster*s are connected by edges in the regularity graph (f). Finally, the nearly co-planarity detection is conducted in each *parallel cluster* and the two dashed primitives in the yellow cluster are detected as nearly co-planar (d), the co-planarity relationships are preserved as node attributes (g).

a pair of adjacent primitives, the regularity operator operates on a regularity graph. The regularity graph contains all the regular relationships in the configuration and is built by progressively detecting nearly parallel, nearly orthogonal and nearly co-planar primitives, Figure 5.21:

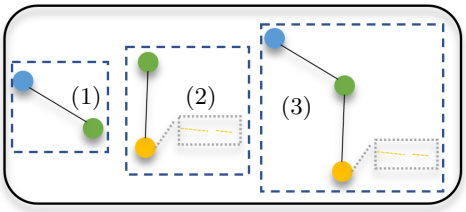
1. **Detection of nearly parallel primitives.** We first detect the nearly parallel primitives and generate *parallel cluster*s. To do this, the support plane of each primitive is first projected onto a unit sphere according to its normal. Then they are grouped into several *parallel cluster*s via mean-shift using the user-defined parallel angle tolerance β as the Gaussian kernel. In each *parallel cluster*, there are either a set of nearly parallel primitives or only one primitive. Note that, a primitive can only belong to one *parallel cluster*. The *parallel cluster*s are seen as the nodes of the regularity graph.

2. **Detection of nearly orthogonal *parallel cluster*s.** We then find

the nearly orthogonal *parallel cluster*s by iteratively checking random pairs of *parallel cluster*s. Two *parallel cluster*s cp_1 and cp_2 are nearly orthogonal if and only if $|n_{cp_1} \cdot n_{cp_2}| < sin(\beta)$ where n_{cp_1} and n_{cp_2} are the average normals of these two *parallel cluster*s. The nearly orthogonal *parallel cluster*s are connected by edges in the regularity graph.

3. **Detection of nearly co-planar primitives.** We finally detect the nearly co-planar primitives in each *parallel cluster*. The primitives of a *parallel cluster* are clustered based on their signed distances to the origin, which is also solved via mean-shift clustering using user-defined coplanar distance tolerance ε_c as Gaussian kernel. The co-planarity relationships are taken as the node attributes in the regularity graph.

The regular relationships are not independent and complicated [SRF+14] such as orthogonal relationships are built based on *parallel cluster*s. Therefore, we cannot individually conduct the regularization of each regular relationship. As a result, we introduce a constraint to simplify the solution space: the regularity operator can only operate on a subgraph induced on the regularity graph by a random node and its adjacent nodes. For instance in the inset, three desired induced subgraphs can be generated from the regularity graph in Figure 5.21 (g): the subgraph (1) is induced by the blue node and its adjacent green node; departing from the yellow node, its adjacent green node is first found before inducing the subgraph (2); the subgraph (3) is induced by the green node and its two adjacent nodes (blue and yellow). And for each induced subgraph, we consider the initial node as its *main node*.



Given a desired induced subgraph, we regularize the primitives in it by repositioning the support planes of the primitives. More precisely, we first adjust the primitives in each node to be exactly parallel. The *main node* of the subgraph is then fixed and the rest nodes are rotated to be exactly orthogonal to it. Finally, the nearly co-planar primitives are moved on the same plane, Figure 5.22.

To guide the order of the regularity operations on the set of induced subgraphs, we update energy function 5.1 by introducing the regularity term.

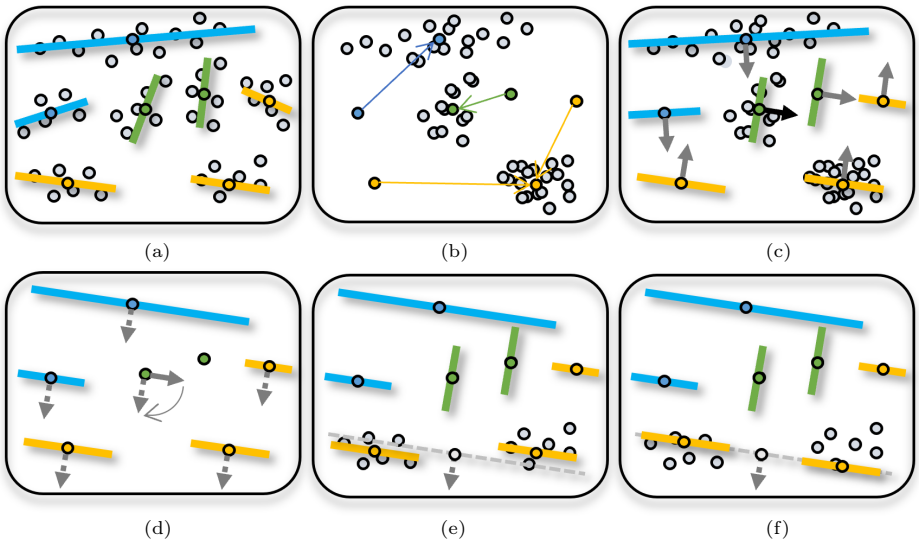


Figure 5.22: Regularity operation conducted on the induced subgraph (3) in the inset on the previous page. The primitives in the nodes (*parallel cluster*s) are first adjusted to be exactly parallel (a), (b) and (c). The centers of mass of corresponding inlier points are first calculated (a), then the inlier points are translated together for each node by moving the centers of mass to a single place (b) and the mean normal for each node is estimated from the translated inlier points using *PCA* (c). Then the exactly parallel primitives are located by the centers of mass before moving and the mean normals (c). We then freeze the *main node* (green one) and rotate the rest nodes so that they are exactly orthogonal to the *main node* (d). Finally, the inlier points of nearly co-planar primitives are collected to calculate the center of mass, which is used to determine the plane with the mean normal of the corresponding node (yellow) (e) and the nearly co-planar primitives are projected onto the plane (f).

Regularity term. To quantify the geometric regularity of a planar shape configuration, we introduce the degrees of freedom as in [OLA16]. Each primitive has three degrees of freedom, two for the orientation and one for the signed distance to the origin. For a set of parallel planar primitives, we count 2 degrees of freedom for their orientation since they are identical. The co-planar primitives are located on the same support plane, therefore, they have the identical signed distances to the origin and we count 1 degree of freedom for all of their signed distances. For two or more sets of orthogonal

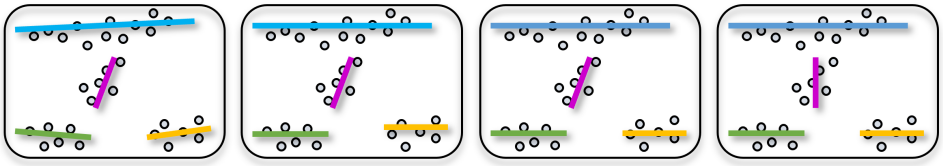


Figure 5.23: Degrees of freedom. In the (a) planar primitive configuration, all the four planar primitives are best fitted to their inlier points and there are not geometric regularity in the configuration. The degrees of freedom of (a) are 12. The blue, green and yellow primitives in the (b) configuration are made parallel by aligning their orientations, the degrees of freedom are then changed to 8. Moving the blue and yellow primitives to the same plane results in configuration (c) decreases the degrees of freedom to 7. In the (d) configuration, the purple primitive is rotated to be orthogonal to the other primitives, and the degrees of freedom are decreased to 6.

primitives, we count 3 degrees of freedom for their orientations. Figure 5.23 illustrates that a configuration has less degrees of freedom when it contains more regular primitives. We measure the geometric regularity of a primitive configuration $\mathbf{x}$ with regularity term U_d whose interval is also $[0, 1]$ same as fidelity, simplicity and completeness terms:

$$U_d(\mathbf{x}) = \frac{DoF(\mathbf{x})}{3n_\sigma} \tag{5.5}$$

where $DoF(\mathbf{X})$ denotes the function for calculating the degrees of freedom of the current configuration as described before. $3n_\sigma$ is the maximum degrees of freedom as n_σ is the maximum number of detectable primitives.

Then the regularity term U_d is added to the energy U 5.1 and weighted by ω_d such that $\omega_f + \omega_s + \omega_c + \omega_d = 1$. Figure 5.24 illustrates the impact of the weight. The energy U is now formed as:

$$U(\mathbf{x}) = \omega_f U_f(\mathbf{x}) + \omega_s U_s(\mathbf{x}) + \omega_c U_c(\mathbf{x}) + \omega_d U_d(\mathbf{x}) \tag{5.6}$$

Same as the local exclusion, insertion, merging and splitting operations, the regularity operations are also guided by a dynamic *priority queue*. However, they conflict with the global transfer operator since exchanging inliers between adjacent primitives can disrupt the geometric regularities in the configuration, which may make the exploration loop infinitely. We, therefore, let the transfer operator skip the regularized primitives P_r after n_r loops,

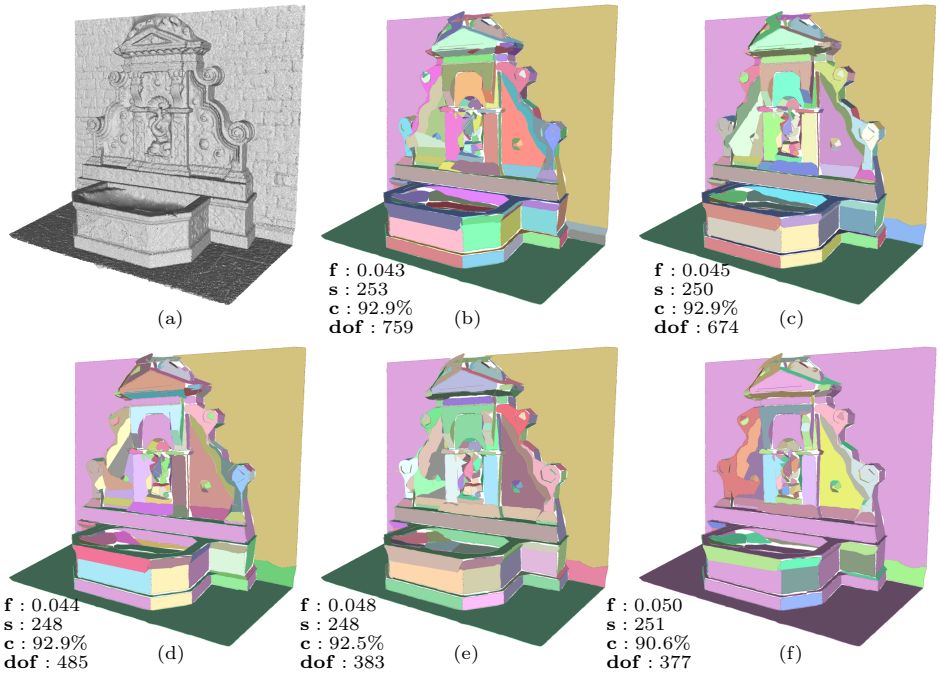


Figure 5.24: Impact of the regularity weight ω_d. Given the point cloud of a fountain (a), more and more regular configurations are produced by increasing the value of ω_d, (b)-(f). Note that the other three weights are fixed and identical. The primitives have the same color if they are parallel, orthogonal or co-planar to each other. **dof** corresponds to the degrees of freedom.

we typically set n_r to 7 in our experiment. The pseudo-code is shown in Algorithm 2.

5.1.4.2 Experiments

In the experiments, we typically set the angle tolerance β to 5 degrees and the coplanar distance tolerance ε_c to $\frac{\varepsilon}{2}$. We denote the proposed shape detection methods with and without regularization by **Regular** and **Original**, respectively. And they are compared only on man-made objects and urban scenes where the geometric regularities should be preserved.

We first tested the extended method on some complex dense point clouds generated from different types of acquisition systems. Our methods are robust and scalable enough to handle these point clouds. Figure 5.25 demonstrates that the extended method can preserve well the geometric regularities

Algorithm 2 Pseudo-code of the exploration mechanism with regularization

Initialize the primitive configuration $\mathbf{x}$
$t_{loop} = 0$
repeat
    if $t_{loop} < n_r$ **then**
        Update $\mathbf{x}$ by the global transfer operator
    else
        Find the regularized primitives P_r
        Update $\mathbf{x} \setminus P_r$ by the global transfer operator
    end if
    Initialize the priority queue Q
    while top operation i of Q decreases energy U **do**
        Update $\mathbf{x}$ by operation i
        Update Q
    end while
    $t_{loop}++$
until no update modifies $\mathbf{x}$ any more

in man-made objects and scenes.

We then do a quantitative comparison on 139 point clouds that are sampled from the CAD models of ABC dataset [KMJ+19], the 139 CAD models are complex and contain at least 200 facets and each CAD is sampled with 100K points. The weights for the four terms in the energy function are identical in **Regular** and the weights for the three terms in the energy function are also identical in **Original**. Table 5.8 shows that the **Regular** can significantly decrease the degrees of freedom and slightly lower the number of detected primitives while sacrificing the fidelity and completeness. Due to the trade-off between the objectives, the sacrificing of fidelity and completeness are unavoidable when geometric regularities are expected to be preserved. However, **Regular** can outperform the region growing method [RVDHV06] in all the four aspects, which demonstrates that our method can finding good planar primitive configurations by seeking high fidelity, high simplicity, high completeness and high regularity simultaneously.

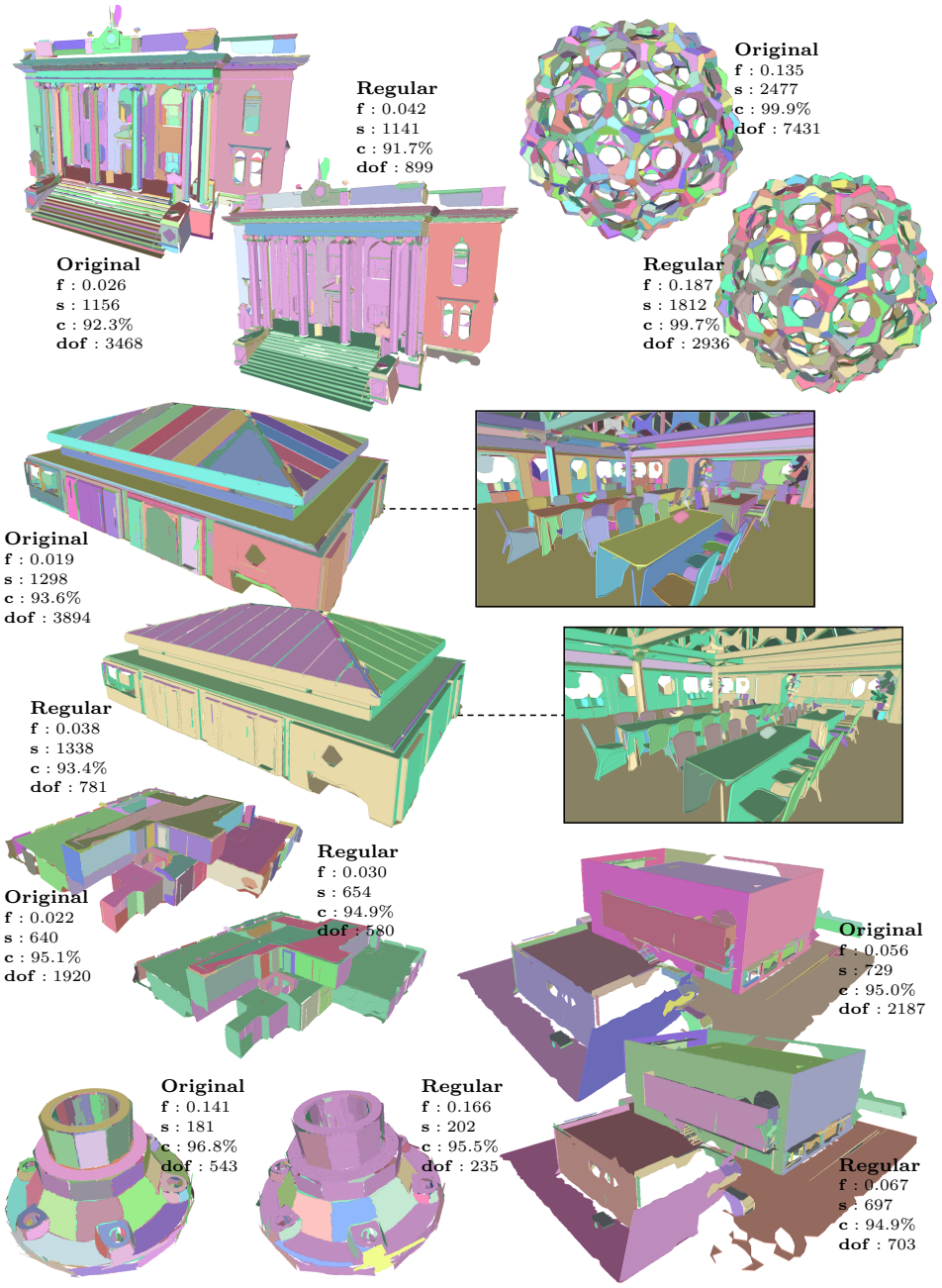


Figure 5.25: Visual comparisons between **Regular** and **Original**. The primitives have the same color if they are parallel, orthogonal or co-planar to each other. **dof** decreased a lot when we take regularity into account.

	Fidelity	Completeness	Simplicity	Degrees of freedom
Original	0.0785	92.86%	174.49	523.47
Regular	0.0898	92.11%	167.45	204.98
RG [RVDHV06]	0.1117	91.56%	197.62	592.86

Table 5.8: Comparison between **Regular**, **Original** and region growing (RG) [RVDHV06] on synthetic point clouds.

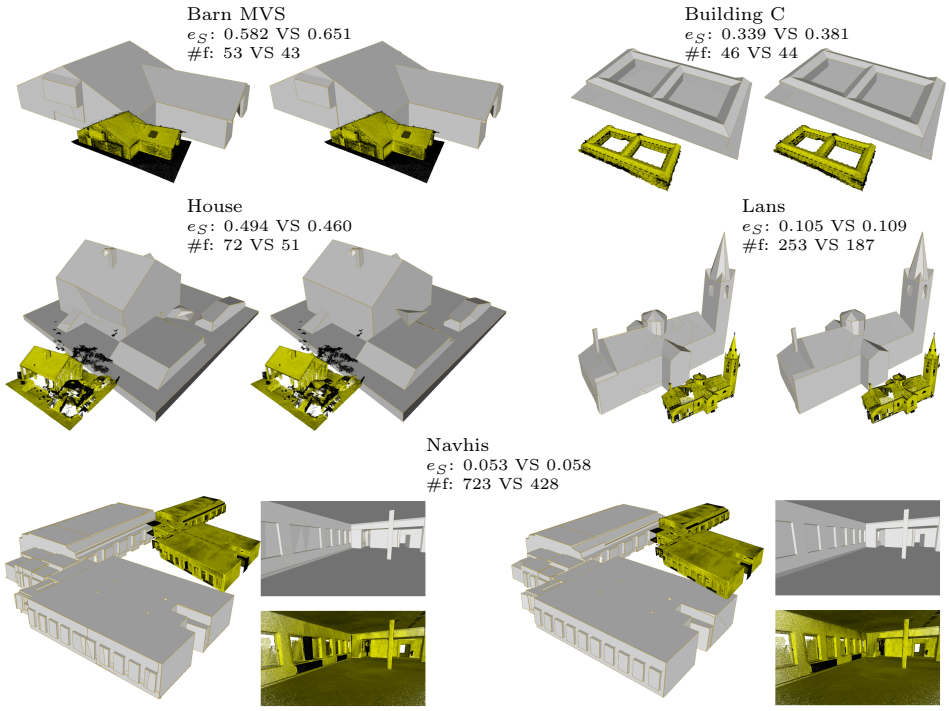


Figure 5.26: Reconstruction of urban objects and scenes. For each pair of compact meshes, the left one and the right one are assembled from the planar primitives generated by **Original** and **Regular**, respectively. e_S and #f refer to the symmetric mean Hausdorff error (in % of the bounding box diagonal) and the number of output facets, respectively. The colored point clouds show the Hausdorff distance distribution (yellow= 0, black$\geq \varepsilon$).

In addition, we also assemble the planar primitives detected by **Regular** or **Original** using KSR. The reconstructed compact meshes are shown in Figure 5.26. We measure the accuracy of the compact mesh by the symmetric mean Hausdorff distance between input point cloud and the output mesh. The simplicity is quantified by the number of polygonal facets of the reconstructed compact mesh. Figure 5.26 demonstrates that the regularized

planar primitives can lead to a more compact mesh where geometric regularities are preserved while sacrificing the accuracy. Particularly, adjusting nearly co-planar primitives to be exactly co-planar can eliminate unnecessary creases in the planar area.

5.2 Surface extraction without normal orientation

Surface extraction is formalized as an energy-minimizing task to determinate the occupancy of each polyhedral cell in KSR [BL20]. To build the energy function, the normal orientation is required, which is not easy to obtain in the real world due to the limitations of acquisition tools. Therefore, a more practical way is required to label the polyhedral cells in the partition only from the coordinates of input points. With the rapid development of deep learning, some implicit neural networks are proposed to estimate the implicit function directly from the point coordinates. Given an estimated implicit function, the energy function can be built by propagating the occupancy information from the estimated implicit function to the polyhedral partition. We, therefore, adapt an efficient and scalable implicit neural network POCO [BM22] in the generate framework (KSR). POCO predicts point-wise latent vectors for each input point before computing latent vectors of the given query points using learning-based interpolation. The query point features are then decoded to occupancy probabilities. In the training phase, the probabilities are used to calculate the loss function with the ground truth of the given query points. In the inference phase, the occupancy probabilities are used to determinate if the corresponding query points are inside or outside of the object that is represented by the input points, Figure 5.27.

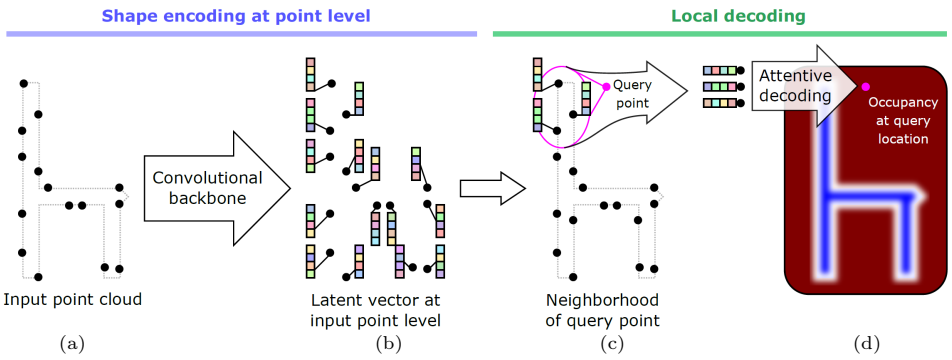
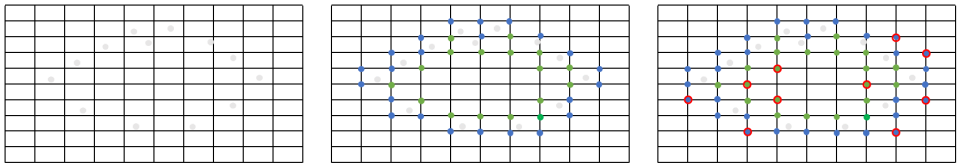


Figure 5.27: Overview of POCO [BM22] during the inference phase. Given a point cloud (a) that represents an object, POCO first extracts point-wise latent vectors (b). For any given query point, its latent vector is generated by interpolating the latent vectors of nearby input points (c). The latent vector is then taken as the input of a decoder to predict the occupancy probability of the query point (d). Image taken from [BM22].



(a) (b) (c)

Figure 5.28: MC-Regro. The bounding box of the input points is discretized using a grid of voxels based on a user-defined parameter *voxel resolution* (10 here) (a). Given another user-defined parameter *dilation size* (1 here), the close endpoints (at a distance of no more than *dilation size* grid steps) are first taken as query points departing from the input points (b). And the occupancy of each query point is predicted, where blue represents empty and green means occupied (b). The occupancy is then iteratively calculated for the rest endpoints that are close (at a distance of no more than *dilation size* grid steps) to both empty and occupied queried points, which have red contours in (c). Only one iteration is carried out here.

5.2.1 Algorithm

The input point cloud is taken as the inputs of KSR and POCO to obtain the kinetic partition and the estimated occupancy function. To extract the occupancy information from the estimated occupancy function, we first collect a set of query points $\mathbf{Q}$ in the vicinity of the input points and predict their occupancy information using the occupancy function estimated by POCO. As for POCO, the set of query points $\mathbf{Q}$ is generated by MC-regro (Marching cube - region growing), Figure 5.28. To propagate the occupancy information of $\mathbf{Q}$ into the kinetic partition, we determine position relationships between $\mathbf{Q}$ and the set of polyhedral cell $\mathbf{C}$ in the kinetic partition. A polyhedral cell gathers the occupancy information from its internal query points. This step is explained in **Query point positioning**. After all, we build an energy function and minimize it using graph cut (**Semantic labeling**).

Query point positioning. To transfer the occupancy information from the set of query points $\mathbf{Q}$ to the kinetic partition, we find all the internal query points for each polyhedral cell in the partition. A polyhedral cell i can be seen as an intersection of a set of half-spaces that are obtained by cutting the 3D space using the support planes of the polygonal facets of i.

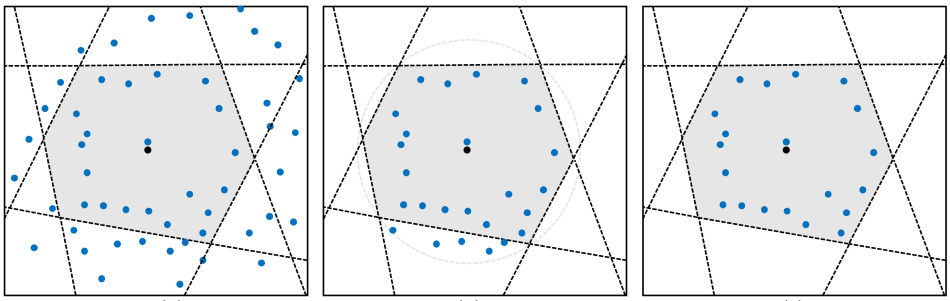
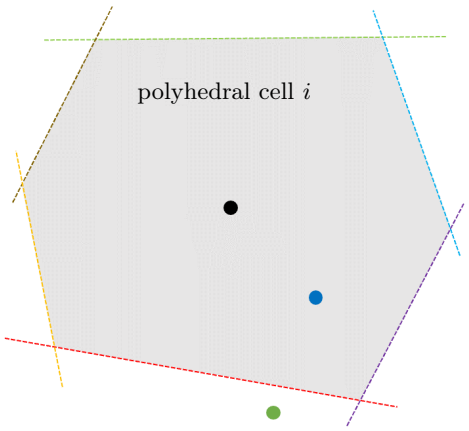


Figure 5.29: Finding the internal query points of a polyhedral cell. Given a set of query points (blue points) and a kinetic partition (dashed line), we want to efficiently find all the internal query points of the polyhedral cell i, represented in grey (a). The query points are first saved in a kd-tree and then the candidate internal query points for the polyhedral cell i are gathered by searching the query points that are located within the bounding sphere (grey dash circle) of i (b). Note that the center of the bounding sphere is located at the barycenter of i. Finally, the internal query points of i are determined from the candidate points, using the barycenter as a reference.



If a reference point that is inside the polyhedral cell i can be discovered, the challenge of determining whether a query point is inside the polyhedral cell i can be converted to determining whether the query point is on the same side of all the support planes of i's facets as the reference point. Thanks to the convexity of the polyhedral cells in the kinetic partition, the reference point can be easily found as the barycenter of each polyhedral cell. In the insert for instance, the black point represents the barycenter of the polyhedral cell i and is seen as the reference point. The dash lines present the support planes of the facets of i. The blue query point is inside the cell i because it is on the same side of all the support planes as the reference point, whereas the green query point is outside i since it is on the other side of the red support plane from the reference point.

Iteratively checking each query point for each polyhedral cell is the simplest way, but it takes a lot of time to position all the internal query points of all the polyhedral cells. We, therefore, proposed a way to first select the candidate points for each polyhedral cell before determining the internal query points from them. To collect the candidate points of a polyhedral cell, the query points are first saved in a kd-tree [TF22] before finding all the query points that are in the bounding sphere of the polyhedral cell, Figure 5.29.

Semantic labeling. We now propagate the occupancy information $\mathbf{O}$ from the query points to the polyhedral cells $\mathbf{C}$ and assign *inside* or *outside* to each polyhedral cell in the kinetic partition. The occupancy of a query point $q \in \mathbf{Q}$ and a polyhedral cell $i \in \mathbf{C}$ are denoted as O_q and x_i, respectively. A two-term energy function is proposed to measure the quality of a set of predicted occupancy $\mathbf{x} = (x_i)_{i \in \mathbf{C}}$:

$$U(\mathbf{x}) = D(\mathbf{x}) + \lambda V(\mathbf{x}) \tag{5.7}$$

where $D(\mathbf{x})$ and $V(\mathbf{x})$ are fidelity and complexity terms, which are living in the interval $[0, 1]$. $\lambda \in [0, 1]$ is a parameter balancing these two terms. The complexity term $V(\mathbf{x})$ measures the complexity of the output surface by its area and a simple surface is preferred by this term, as in [BL20].

The fidelity term $D(\mathbf{x})$ measures the consistency between the occupancy $\mathbf{x}$ assigned to the polyhedral cells and the occupancy of the query points estimated by POCO, which is a sum of local consistency measures over each query point q:

$$D(\mathbf{x}) = \frac{1}{|\mathbf{Q}|} \sum_{i \in \mathbf{C}} \sum_{q \in Q_i} d(x_i, O_q) \tag{5.8}$$

where Q_i is the set of internal query points of the polyhedral cell i, $|\mathbf{Q}|$ is the number of query points and $d(x_i, O_q)$ is a function measuring the consistency between the occupancy x_i of polyhedral cell i and the occupancy O_q of query point q:

$$d(x_i, O_q) = \begin{cases} 1 & \text{if } x_i \neq O_q \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

5.2.2 Experiments

KSR is implemented in C++ and POCO is implemented in Python. The pre-trained POCO network (trained using 4950 CAD models in the ABC

	symmetric mean Hausdorff distance	# facets
KSR [BL20]	0.31%	221
KS-PO	0.36%	216

Table 5.9: Quantitative comparison with KSR. The symmetric mean Hausdorff distance is in % of the bounding box diagonal and # facets presents the number of the polygonal facets of the reconstructed compact mesh.

dataset) is used. To collect query points, the two parameters *voxel resolution* and *dilation size* are set to 256^3 and 2 in MC-Regro, respectively. We denote the combination of KSR and POCO by KS-PO.

Metrics. Two criteria are employed to evaluate the quality of the reconstructed compact meshes. We measure the fidelity by the symmetric mean Hausdorff distance between the input point cloud and the output mesh. The simplicity is quantified by the number of polygonal facets of the reconstructed compact mesh.

We first quantitatively compared KSR and KS-PO on 139 point clouds, which are sampled from the CAD models of ABC dataset [KMJ+19]. The 139 CAD models are complex and contain at least 200 facets and each CAD is sampled with 100K points. Note that the planar shape detection used in KSR and KS-PO are the same. Table 5.9 shows that KS-PO has larger symmetric mean Hausdorff distance and fewer polygonal facets than KSR, it is hard to definitively state which one is better based just on the two measurements. However, the two metrics of the results produced by these two reconstruction methods are essentially the same and KS-PO does not have the limitation of normal orientation. Hence, KS-PO is more practical on such synthetic data.

We then qualitatively compare them on different freeform and man-made object that are generated from different types of acquisition systems, Figure 5.30. Same as KSR, KS-PO is also scalable enough to handle large-scale point clouds. However, KS-PO can not generate accurate compact meshes as good as KSR. Some details are ignored by KS-PO since POCO is not robust enough for real-world data. In addition, POCO can only produce results when the input point clouds describe watertight objects or scenes due to the limitation of the occupancy function. As a result, KS-PO can not work well on the point clouds that contain a large amount of missing data, Figure 5.31.

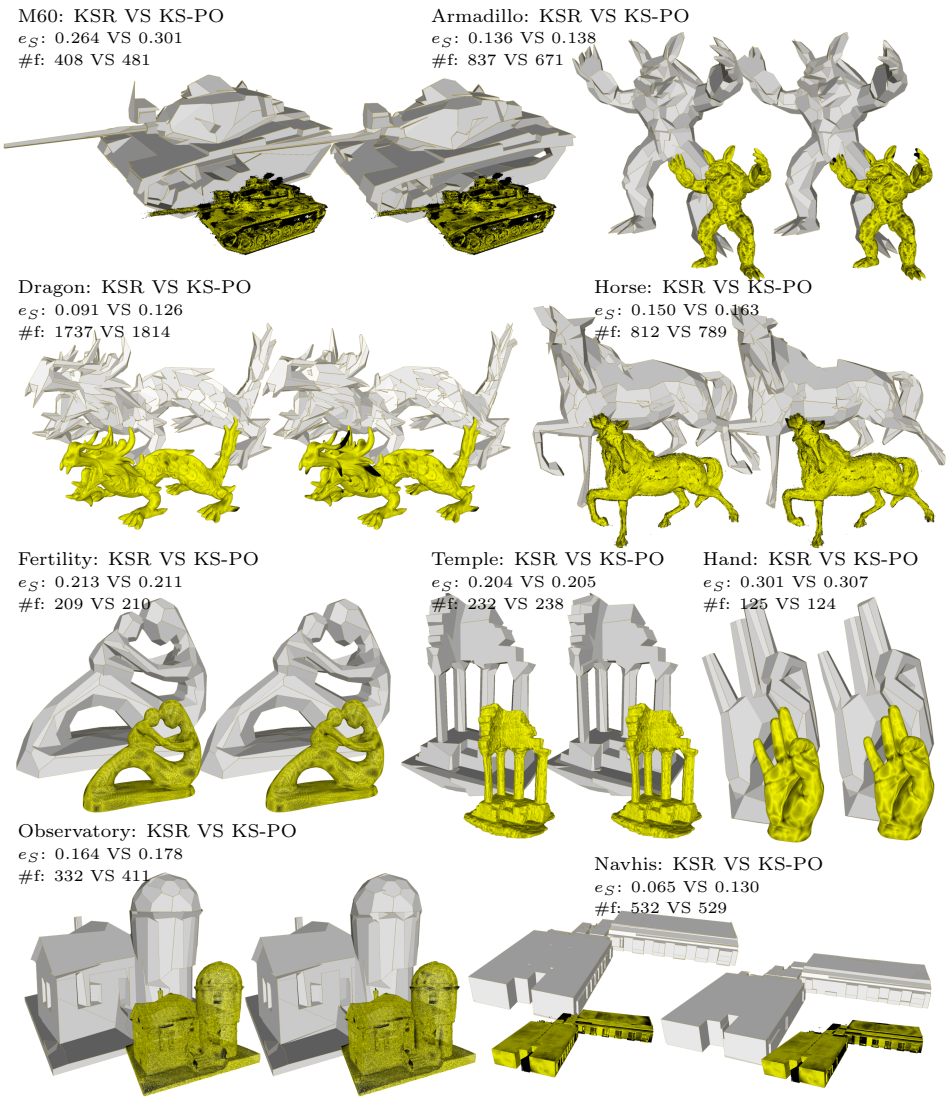


Figure 5.30: Visual comparison with KSR. In each pair of result, the left one is the reconstructed result of KSR and the right one is reconstructed by KS-PO. The colored point clouds show the geometric error distribution (yellow=0, black$\geq \varepsilon$). e_S, #f refer to the symmetric mean Hausdorff error (in % of the bounding box diagonal) and the number of output facets.

5.3 Conclusion

To efficiently reconstruct compact meshes, we first proposed an algorithm for fitting planar primitives to unorganized point clouds. The key contribution of this work relies upon the design and efficient implementation of

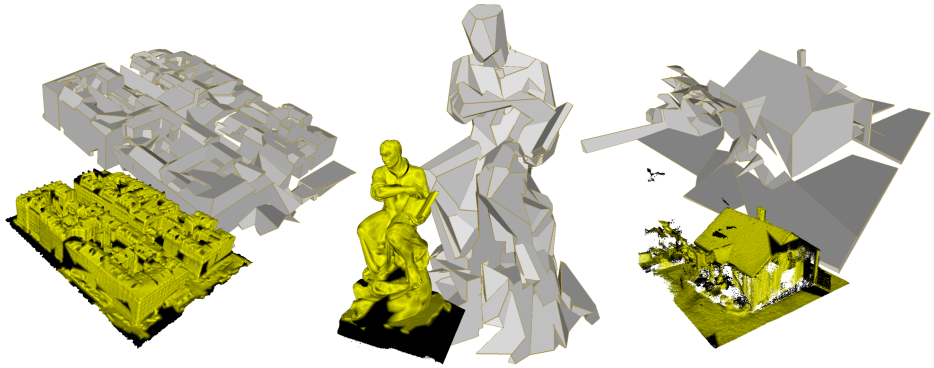


Figure 5.31: Failure cases of KS-PO. When there are large missing parts in the input point clouds, KS-PO can not deliver well reconstructed compact meshes.

an exploration mechanism that seeks configurations with high fidelity, high simplicity and high completeness simultaneously. In addition, the algorithm can be extended such that regularity can also be taken into account. Inspired by geometry processing techniques, this mechanism delivers high quality results that outclass those obtained with traditional methods and recent deep learning models. We also demonstrated the efficiency and the robustness of our algorithm on a variety of objects and scenes in terms of size, complexity and acquisition characteristics. To make the reconstruction method more practical and free from the constraints of the requirement for normal orientation, a new compact mesh reconstruction pipeline is proposed based on the combination of a reconstruction method (KSR [BL20]) and an implicit neural network (POCO [BM22]). Thanks to the powerful learning capacity of the neural network, occupancy information can be directly estimated from the point coordinates. Our experiments demonstrated the practicality of the combined pipeline.

Chapter 6

Conclusion and Perspectives

6.1 Conclusion

This thesis addressed the problem of obtaining compact and accurate 3D virtual models from the common, readily available data in urban contexts, which involves repairing defect-laden urban models and compact meshes reconstruction from point clouds. In terms of repairing, a generic and global repairing method was presented to correct geometric errors in common urban models. On the reconstruction side, we enhanced the planar shape detection and surface extraction components of an efficient and scalable compact mesh reconstruction method KSR [BL20], making it more practical to produce compact meshes of high quality.

3D model repairing. In Section 4, we proposed a repairing method that allows generic and global correction of geometric errors in both facet-based and volume-based urban models by using Kinetic data structures. Comparison with traditional mesh repairing methods demonstrated that our repairing method is global and object-aligned, which can ensure the validation of output and preserve the sharp features such as corners and creases. Experiments on various CityGML models and large IFC models showed the adaptability and scalability of our repairing method. Furthermore, experiments on different types of errors, such as gaps, overlaps between volumes, self-intersection of facets and vertex misalignment, demonstrated that our method is robust to various types of geometric errors. Our method has also been shown to compete well against the existing urban model repairing methods. In addition, a confidence map of the predicted semantics in ambiguous cases can also be provided by our method, which can simplify optional user interactions. Moreover, we designed a user-interactive GUI that enables users to interactively correct some failure instances of our repairing procedure, producing ideal and user-desired results.

Compact mesh reconstruction. In Section 5.1, we first presented an algorithm for detecting good planar shape configurations from point clouds. The characteristics of a good planar shape configuration was defined before proposing a natural and simple energy function. Guided by the energy function, five geometric operators are iteratively conducted to explore a mixed discrete-and-continuous configuration space. Experiments on CAD-sampled, Laser scanned and MVS point clouds demonstrated that our planar shape detection method is robust in terms of acquisition tools; experiments on freeform objects and urban scenes showed the flexibility and scalability of our method. Meanwhile, it has been shown to compete well against existing incremental mechanisms and neural network approaches. Our method is extensible, i.e. other objectives can be taken into account as long as they can be formalized as new terms in the energy function and relevant new operators can be devised. We verified the extensibility by introducing geometric regularity as a new objective in Section 5.1.4, where degrees of freedom are leveraged to measure the regularity of planar shape configurations and the regularity operators are proposed based on a pre-detected regularity graph. In addition, we replaced the Region Growing method in the original KSR with our method, the updated KSR outperforms the state-of-the-art compact mesh reconstruction methods [BL20, NW17, CTZ20].

We secondly enhanced the surface extraction component of KSR by using an implicit neural network POCO [BM22] in Section 5.2. This improves the practicability of the reconstruction pipeline. The combination of explicit and implicit methods benefits from both the structure-preserving of the kinetic data structure and the powerful learning capability of the neural network. Experiments have shown that the combined pipeline can produce compact meshes with roughly the same quality as the results of KSR on synthetic data while does not have the restriction that normal orientation is necessary.

6.2 Perspectives

This thesis only constitutes a tiny step toward reconstruction and repairing of 3D urban models. Our methods suffer from some weaknesses. As detailed in Section 5, the shape detection method requires long processing time for handling the large-scale point clouds and is influenced by the initial configuration. The 3D model repairing method can not automatically handle the

ambiguous semantic errors as discussed in Section 4. These weaknesses call for several future research that needs to be investigated.

Parallel explorations. Our planar shape detection method sequentially conducts the operations, which results in lengthy processing times on massive point clouds, e.g. urban scenes. Inspired by the two-level hierarchical partitioning used in [CLP10], separating a large point cloud into several semantically consistent parts and then optimizing the planar shape configuration of each part in parallel would be a good way to reduce the running time. A semantic point cloud segmentation method and a new operator for combining the configuration of each part are therefore desired to be designed.

Generalization to more complex primitives. Our shape detection method can only fit planar shapes, a natural extension of this work would be to accommodate a wider range of shapes. Spheres, cylinders, cones and B-spline surfaces can be introduced to enrich the shape types and lead to simpler and more accurate configurations. Designing a unified energy function and the operators that are applicable to all shapes is the key challenge.

Robustness to initialization. As shown in Figure 5.3, poor initializations can easily lead to badly optimized configurations. The reason is that our exploration mechanism is not designed to find the global optimal solution and the configuration may get stuck at a locally optimal solution. To improve the robustness to initialization, one possible solution is to design a new operator to disturb the configuration and skip over the local optimal solution; another way could be to optimize the configuration with the help of reinforcement learning.

Semantic understanding. In urban models, there exist semantic errors in addition to geometric errors. Our repairing method can not automatically deal with the ambiguous cases in IFC models and needs user interaction. A graph neural network could be a good choice for predicting the semantic class of each polyhedral cell in the kinetic partitioning, while training the network requires an enriched training dataset, which is not yet available but can be generated using the designed user-interactive GUI in Section 4.

Multi-scale Urban Digital Twin (UDT). An ideal UDT can control and manage a city in both city-scale and building-scale. Therefore, creating a multi-scale UDT is a good future research direction. One way could be creating an integration of BIM and GIS [LWW+17]. However, some challenges still need to be addressed, such as geometric and topological issues when converting data with different forms [OBD+17].

Towards an alternative representation of point clouds. The point cloud is commonly used as an intermediate representation when translating geometric information from 2D (images) to 3D (meshes). However, some important information, such as neighbor information, is lost while converting between images and point clouds. Therefore, I believe that it is important to design better representations that can gather more information from the images and can deliver more accurate meshes. For instance, NeRF [MST+20] can estimate a radiance field from the input images, which could be an alternative representation to the point cloud [MHS+22].

Combination of traditional geometric processing and deep learning. Traditional geometric processing and deep learning have advantages in different applications. For instance, deep learning is good at segmentation and detection while geometric processing is good at generation and reconstruction. In my opinion, designing end-to-end deep learning methods to deal with all the geometric problems shall be inefficient, and combining the traditional geometric processing and deep learning in an efficient manner would be a more interesting direction for future research. For instance, [LLM20] extracts and vectorizes objects with polygons in images based on rough semantic probability maps generated by deep learning method [LY16]. It delivers better results than end-to-end RNN-based methods [CKUF17, LWL19].

Publications

This thesis is supported by the following publications:

- Mulin Yu, Florent Lafarge. Finding Good Configurations of Planar Primitives in Unorganized Point Clouds. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022.
- Mulin Yu, Florent Lafarge, Sven Oesau, Bruno Hilaire. Repairing geometric errors in 3D urban models with kinetic data structures. ISPRS Journal of Photogrammetry and Remote Sensing 192 (2022): 315-326.

Additional preprint not within the scope of this thesis:

- Tong Zhao, Mulin Yu, Florent Lafarge, Pierre Alliez. Sharp Feature Consolidation from Raw 3D Point Clouds via Displacement Learning. Preprint.

Bibliography

[ASF+13] Murat Arikan, Michael Schwarzler, Simon Flory, Michael Wimmer, and Stefan Maierhofer. O-Snap: Optimization-Based Snapping for Modeling Architecture. *Trans. on Graphics*, 32(1), 2013. (Cited on pages 23 and 24.)

[Att10] Marco Attene. A lightweight approach to repairing digitized polygon meshes. *The Visual Computer*, 26(11), 2010. (Cited on pages 13, 45 and 48.)

[AWW+14] Nazmul Alam, Detlev Wagner, Mark Wewetzer, Julius von Falkenhausen, Volker Coors, and Margitta Pries. Towards automatic validation and healing of CityGML models for geometric and semantic consistency. In *Innovations in 3D Geo-Information Sciences*. 2014. (Cited on page 16.)

[BBN+20] Tolga Birdal, Benjamin Busam, Nassir Navab, Slobodan Ilic, and Peter Sturm. Generic Primitive Detection in Point Clouds Using Novel Minimal Quadric Fits. *TPAMI*, 42(6), 2020. (Cited on page 20.)

[BdLGM14] Alexandre Boulch, Martin de La Gorce, and Renaud Marlet. Piecewise-planar 3D reconstruction with edge and corner regularization. In *Computer Graphics Forum*, volume 33, 2014. (Cited on page 24.)

[BELN11] Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas Nüchter. The 3D hough transform for plane detection in point clouds: A review and a new accumulator design. *3D Research*, 2(2), 2011. (Cited on page 20.)

[BK97] Gill Barequet and Subodh Kumar. Repairing CAD models. In *Proceedings. Visualization*, 1997. (Cited on page 13.)

[BK05] Stephan Bischoff and Leif Kobbelt. Structure preserving CAD model repair. In *Computer Graphics Forum*, volume 24, 2005. (Cited on page 15.)

[BKP+10] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *"Polygon Mesh Processing"*. AK Peters, 2010. (Cited on page 59.)

[BL20] Jean-Philippe Bauchet and Florent Lafarge. Kinetic Shape Reconstruction. *Trans. on Graphics*, 39(5), 2020. (Cited on pages 25, 27, 28, 32, 37, 50, 55, 65, 69, 71, 72, 95, 98, 99, 101, 103 and 104.)

[BLD+16] Filip Biljecki, Hugo Ledoux, Xin Du, Jantien Stoter, Kean Huat Soon, and VHS Khoo. The most common geometric and semantic errors in CityGML datasets. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 4, 2016. (Cited on pages 6, 15 and 42.)

[BLS16] Filip Biljecki, Hugo Ledoux, and Jantien Stoter. An improved LOD specification for 3D building models. *Computers, Environment and Urban Systems*, 59, 2016. (Cited on page 4.)

[BM22] Alexandre Boulch and Renaud Marlet. POCO: Point convolution for surface reconstruction. In *CVPR*, 2022. (Cited on pages 17, 30, 95, 101 and 104.)

[BS95] Gill Barequet and Micha Sharir. Filling gaps in the boundary of a polyhedron. *Computer Aided Geometric Design*, 12(2), 1995. (Cited on page 13.)

[BSL+15] Filip Biljecki, Jantien Stoter, Hugo Ledoux, Sisi Zlatanova, and Arzu Çöltekin. Applications of 3D city models: State of the art review. *ISPRS International Journal of Geo-Information*, 4(4), 2015. (Cited on page 1.)

[Bui] BuildingSMART Polska. BIMvisio. `https://bimvision.eu/`. (Cited on page 2.)

[CAK12] Marcel Campen, Marco Attene, and Leif Kobbelt. A Practical Guide to Polygon Mesh Repairing. In *Eurographics (Tutorials)*, 2012. (Cited on page 13.)

[CAPM20] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion. In *CVPR*, 2020. (Cited on page 17.)

[CC08] Jie Chen and Baoquan Chen. Architectural Modeling from Sparsely Scanned Range Data. *IJCV*, 78(2-3), 2008. (Cited on pages 20 and 23.)

[CDF+17] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. *3DV*, 2017. (Cited on page 8.)

[Ces] Cesium Engineer Team. CESIUM (Open Source 3D rendering library). `https://github.com/CesiumGS/cesium`. (Cited on page 3.)

[CFG+15] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. Technical report, 2015. (Cited on pages 70 and 72.)

[Cha16] Kang-Tsung Chang. Geographic information system. *International Encyclopedia of Geography: People, the Earth, Environment and Technology: People, the Earth, Environment and Technology*, 2016. (Cited on page 4.)

[CHXS20] Rui Chen, Songfang Han, Jing Xu, and Hao Su. Visibility-Aware Point-Based Multi-View Stereo Network. *TPAMI*, 43(10), 2020. (Cited on page 9.)

[CKUF17] Lluis Castrejon, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Annotating object instances with a Polygon-RNN. In *CVPR*, 2017. (Cited on page 106.)

[CLP10] Anne-Laure Chauve, Patrick Labatut, and Jean-Philippe Pons. Robust Piecewise-Planar 3D Reconstruction and Completion from Large-Scale Unstructured Point Data. In *CVPR*, 2010. (Cited on pages 24, 25, 72, 73, 75 and 105.)

[CMPM20] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. Neural unsigned distance fields for implicit function learning. In *NIPS*, volume 33, 2020. (Cited on page 17.)

[CSAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. In *ACM SIGGRAPH*, 2004. (Cited on pages 18, 19, 21 and 72.)

[CSaLM13] Desai Chen, Pitchaya Sitthi-amorn, Justin T Lan, and Wojciech Matusik. Computing and fabricating multiplanar models. In *Computer graphics forum*, volume 32, 2013. (Cited on page 19.)

[CT11] Fatih Calakli and Gabriel Taubin. SSD: Smooth signed distance surface reconstruction. In *Computer Graphics Forum*, volume 30, 2011. (Cited on page 17.)

[CTZ20] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. BSP-Net: Generating Compact Meshes via Binary Space Partitioning. In *CVPR*, 2020. (Cited on pages 25, 55, 70, 72 and 104.)

[CY00] James M. Coughlan and Alan L. Yuille. The Manhattan world assumption: Regularities in scene statistics which enable Bayesian inference. In *NIPS*, 2000. (Cited on page 22.)

[DDVM14] Abdoulaye Abou Diakité, Guillaume Damiand, and Dirk Van Maercke. Topological reconstruction of complex 3D buildings and automatic extraction of levels of detail. In *Eurographics Workshop on Urban Data Modelling and Visualisation*, 2014. (Cited on pages 16 and 32.)

[Dep16] Department of Information Technology and Telecommunications (DOITT). The NYC 3-D Building Massing Model, 2016. https://github.com/CityOfNewYork/nyc-geo-metadata/blob/master/Metadata/. (Cited on page 3.)

[DI15] Bertram Drost and Slobodan Ilic. Local Hough Transform for 3D Primitive Detection. In *3DV*, 2015. (Cited on page 20.)

[DL14] Guillaume Damiand and Pascal Lienhardt. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and*

Image Processing. A K Peters/CRC Press, 2014. (Cited on page 32.)

[EGO+20] Philipp Erler, Paul Guerrero, Stefan Ohrhallinger, Niloy J Mitra, and Michael Wimmer. Points2surf learning implicit surfaces from point clouds. In *ECCV*, 2020. (Cited on page 17.)

[FL20] Hao Fang and Florent Lafarge. Connect-and-Slice: an hybrid approach for reconstructing 3D objects. In *CVPR*, 2020. (Cited on page 26.)

[FLD18] Hao Fang, Florent Lafarge, and Mathieu Desbrun. Planar Shape Detection at Structural Scales. In *CVPR*, 2018. (Cited on page 22.)

[GDJY19] Jianwei Guo, Fan Ding, Xiaohong Jia, and Dong-Ming Yan. Automatic and high-quality surface mesh generation for CAD models. *Computer-Aided Design*, 109, 2019. (Cited on page 14.)

[GFZ+20] Xiaodong Gu, Zhiwen Fan, Siyu Zhu, Zuozhuo Dai, Feitong Tan, and Ping Tan. Cascade Cost Volume for High-Resolution Multi-View Stereo and Stereo Matching. In *CVPR*, 2020. (Cited on page 9.)

[GH97] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997. (Cited on pages 18 and 72.)

[GKUP11] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. BlenSor: Blender sensor simulation toolbox. In *International Symposium on Visual Computing*, 2011. (Cited on page 9.)

[GL12] Groger Gerhard and Plumer Lutz. CityGML - Interoperable semantic 3D city models. *Journal of Photogrammetry and Remote Sensing*, 71, 2012. (Cited on pages 4 and 31.)

[GTLH01] André Guéziec, Gabriel Taubin, Francis Lazarus, and B Hom. Cutting and stitching: Converting sets of polygons to manifold

surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 7(2), 2001. (Cited on page 13.)

[Gui04] Leonidas J. Guibas. Kinetic data structures. In *Handbook of Data Structures and Applications*, 2004. (Cited on page 37.)

[Han21] Hanover City, Department of Planning and Urban Development, Geoinformation. City model Hannover CityGML LoD2, 2021. https://opengeodata.hannover-stadt.de/Stadtmodell_Hannover_CityGML_LoD2.zip. (Cited on page 43.)

[HMFB18] Thomas Holzmann, Michael Maurer, Friedrich Fraundorfer, and Horst Bischof. Semantically Aware Urban 3D Reconstruction with Plane-Based Regularization. In *ECCV*, 2018. (Cited on page 55.)

[Hop99] Hugues Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *Proceedings Visualization*, 1999. (Cited on page 18.)

[HZS21] Jingwei Huang, Yanfeng Zhang, and Mingwei Sun. PrimitiveNet: Primitive Instance Segmentation with Local Primitive Embedding under Adversarial Metric. In *ICCV*, 2021. (Cited on page 22.)

[IB12] Hossam Isack and Yuri Boykov. Energy-based Geometric Multi-Model Fitting. *IJCV*, 97(2), 2012. (Cited on pages 21 and 56.)

[ISO03] ISO ISO. 19107: 2003: Geographic information—Spatial schema. *International Organization for Standardization*, 90, 2003. (Cited on page 4.)

[ISO04] ISO ISO. 10303-11: 2004 industrial automation systems and integration– product data representation and exchange–part 11: Description methods: The express language reference manual. *ISO/TC*, 184, 2004. (Cited on page 2.)

[JJ14] Farzad Jalaei and Ahmad Jrade. Integrating building information modeling (BIM) and energy analysis tools with green

building certification system to conceptually design sustainable buildings. *J. Inf. Technol. Constr.*, 19, 2014. (Cited on page 2.)

[JSM+20] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. Local implicit grid representations for 3D scenes. In *CVPR*, 2020. (Cited on page 17.)

[Ju04] Tao Ju. Robust repair of polygonal models. *Trans. on Graphics*, 23(3), 2004. (Cited on pages 45 and 48.)

[Kae15] Michael Kaess. Simultaneous localization and mapping with infinite planes. In *ICRA*, 2015. (Cited on page 55.)

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006. (Cited on page 17.)

[KCK18] Pyojin Kim, Brian Coltin, and H Jin Kim. Linear RGB-D SLAM for planar environments. In *ECCV*, 2018. (Cited on page 55.)

[KH13] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *Trans. on Graphics*, 32(3), 2013. (Cited on page 17.)

[KMJ+19] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. ABC: A Big CAD Model Dataset For Geometric Deep Learning. In *CVPR*, 2019. (Cited on pages 9, 22, 65, 66, 91 and 99.)

[KNO+20] Thomas Krijnen, Francesca Noardo, Ken Arroyo Ohori, Hugo Ledoux, and Jantien Stoter. Validation and inference of geometrical relationships in ifc. In *Proceedings of the 37th International Conference of CIB W*, volume 78, 2020. (Cited on page 15.)

[KPZK17] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and Temples: Benchmarking Large-Scale Scene

Reconstruction. *Trans. on Graphics*, 36(4), 2017. (Cited on pages 8 and 65.)

[KYZB18] Adrien Kaiser, Jose Alonso Ybanez Zepeda, and Tamy Boubekeur. A Survey of Simple Geometric Primitives Detection Methods for Captured 3D Data. *Computer Graphics Forum*, 37, 2018. (Cited on page 55.)

[LA13] Florent Lafarge and Pierre Alliez. Surface reconstruction through point set structuring. *Computer Graphics Forum*, 32, 2013. (Cited on page 24.)

[LC87] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph*, 21(4), 1987. (Cited on page 17.)

[Led18] Hugo Ledoux. val3dity: validation of 3D GIS primitives according to the international standards. *Open Geospatial Data, Software and Standards*, 3(1), 2018. (Cited on page 43.)

[LGP+21] Shi-Lin Liu, Hao-Xiang Guo, Hao Pan, Peng-Shuai Wang, Xin Tong, and Yang Liu. Deep implicit moving least-squares functions for 3D reconstruction. In *CVPR*, 2021. (Cited on page 17.)

[LGR15] GN Lilis, GI Giannakis, and DV Rovas. Detection and semi-automatic correction of geometric inaccuracies in IFC files. In *14th International Conference of IBPSA-Building Simulation 2015, BS 2015, Conference Proceedings*, 2015. (Cited on page 15.)

[Li,19] Li, Lingxiao and Sung, Minhyuk and Dubrovina, Anastasia and Yi, Li and Guibas, Leonidas J. Supervised fitting of geometric primitives to 3d point clouds. In *CVPR*, 2019. (Cited on pages 21, 56, 65, 66, 67 and 68.)

[Lin00] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000. (Cited on page 18.)

[LKG+19] Chen Liu, Kihwan Kim, Jinwei Gu, Yasutaka Furukawa, and Jan Kautz. PlaneRCNN: 3D Plane Detection and Reconstruction from a Single Image. In *CVPR*, 2019. (Cited on page 22.)

[LL21] Muxingzi Li and Florent Lafarge. Planar Shape Based Registration for Multi-modal Geometry. In *BMVC*, 2021. (Cited on page 55.)

[LLM20] Muxingzi Li, Florent Lafarge, and Renaud Marlet. Approximating shapes in images with low-complexity polygons. In *CVPR*, 2020. (Cited on page 106.)

[Llo82] Stuart Lloyd. Least squares quantization in PCM. *Trans. on Information Theory*, 28(2), 1982. (Cited on page 21.)

[LMBM20] Thibault Lejemble, Claudio Mura, Loïc Barthe, and Nicolas Mellado. Persistence Analysis of Multi-scale Planar Structure Graph in Point Clouds. *Computer Graphics Forum*, 39(2), 2020. (Cited on page 22.)

[LRC+03] David Luebke, Martin Reddy, Jonathan D Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of detail for 3D graphics*. Morgan Kaufmann, 2003. (Cited on page 17.)

[LSC+21] Eric-Tuan Lê, Minhyuk Sung, Duygu Ceylan, Radomir Mech, Tamy Boubekeur, and Niloy J. Mitra. CPFN: Cascaded Primitive Fitting Networks for High-Resolution Point Clouds. In *ICCV*, 2021. (Cited on pages 22 and 56.)

[LT98] Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. In *Proceedings Visualization*. IEEE, 1998. (Cited on page 18.)

[LWC+11] Yangyan Li, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J Mitra. Globfit: Consistently fitting primitives by discovering global relations. *Trans. on Graphics*, 2011. (Cited on page 22.)

[LWL19] Zuoyue Li, Jan Dirk Wegner, and Aurélien Lucchi. Topological map extraction from overhead images. In *ICCV*, 2019. (Cited on page 106.)

[LWW+17] Xin Liu, Xiangyu Wang, Graeme Wright, Jack CP Cheng, Xiao Li, and Rui Liu. A state-of-the-art review on the integration of Building Information Modeling (BIM) and Geographic Information System (GIS). *ISPRS International Journal of Geo-Information*, 6(2), 2017. (Cited on page 106.)

[LY16] Guanbin Li and Yizhou Yu. Deep contrast learning for salient object detection. In *CVPR*, 2016. (Cited on page 106.)

[Met15] Lyon Metropolis. 3D textured model of the town of La Tour-de-Salvagny, 2015. https://download.data.grandlyon.com/files/grandlyon/localisation/bati3d/LA_TOUR_DE_SALVAGNY_2015.zip. (Cited on page 43.)

[MHS+22] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting Triangular 3D Models, Materials, and Lighting From Images. In *CVPR*, 2022. (Cited on page 106.)

[MLM01] David Marshall, Gabor Lukacs, and Ralph Martin. Robust Segmentation of Primitives from Range Data in the Presence of Geometric Degeneracy. *TPAMI*, 23(3), 2001. (Cited on page 20.)

[MMBM15] Aron Monszpart, Nicolas Mellado, Gabriel J Brostow, and Niloy J Mitra. RAPter: rebuilding man-made scenes with regular arrangements of planes. *Trans. on Graphics*, 34(4), 2015. (Cited on pages 21, 22 and 56.)

[MON+19] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*, 2019. (Cited on page 17.)

[MPR12] Emilie Marchandise, Cécile Piret, and J-F Remacle. CAD and mesh repair with radial basis functions. *Journal of Computational Physics*, 231(5), 2012. (Cited on page 14.)

[MST+20] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. (Cited on page 106.)

[MW99] Andrey A Mezentsev and Thomas Woehler. Methods and Algorithms of Automated CAD Repair for Incremental Surface Meshing. In *IMR*, 1999. (Cited on page 13.)

[NT03] Fakir S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2), 2003. (Cited on page 14.)

[NW17] Liangliang Nan and Peter Wonka. PolyFit: Polygonal surface reconstruction from point clouds. In *ICCV*, 2017. (Cited on pages 25, 70, 72, 73, 75 and 104.)

[OBD+17] Ken Arroyo Ohori, Filip Biljecki, Abdoulaye Diakité, Thomas Krijnen, Hugo Ledoux, and Jantien Stoter. Towards an integration of GIS and BIM data: What are the geometric and topological issues. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 4, 2017. (Cited on pages 2, 3, 15 and 106.)

[OLA14] Sven Oesau, Florent Lafarge, and Pierre Alliez. Indoor scene reconstruction using feature sensitive primitive extraction and graph-cut. *ISPRS journal of photogrammetry and remote sensing*, 90, 2014. (Cited on page 24.)

[OLA16] Sven Oesau, Florent Lafarge, and Pierre Alliez. Planar shape detection and regularization in tandem. *Computer Graphics Forum*, 35(1), 2016. (Cited on pages 22, 56 and 88.)

[OVJ+21] Sven Oesau, Yannick Verdie, Clément Jamin, Pierre Alliez, Florent Lafarge, Simon Giraudot, Thien Hoang, and Dmitry Anisimov. Point Set Shape Detection. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.3 edition, 2021. (Cited on pages 20, 65, 66 and 68.)

[PCYS12] Trung-Thanh Pham, Tat-Jun Chin, Jin Yu, and David Suter. The Random Cluster Model for robust geometric fitting. In *CVPR*, 2012. (Cited on page 21.)

[PERW16] Trung T. Pham, Markus Eich, Ian Reid, and Gordon Wyeth. Geometrically Consistent Plane Extraction for Dense Indoor 3D Maps Segmentation. In *IROS*, 2016. (Cited on pages 21 and 56.)

[PFS+19] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019. (Cited on page 17.)

[PMMB22] Abhinesh Prabhakaran, Abdul-Majeed Mahamadu, Lamine Mahdjoubi, and Pawel Boguslawski. BIM-based immersive collaborative environment for furniture, fixture and equipment design. *Automation in Construction*, 142, 2022. (Cited on page 2.)

[PNM+20] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *ECCV*, 2020. (Cited on page 17.)

[PSMA16] Emiliano Pérez, Santiago Salamanca, Pilar Merchán, and Antonio Adán. A comparison of hole-filling methods in 3d. *International Journal of Applied Mathematics and Computer Science*, 26(4), 2016. (Cited on page 13.)

[QF20] Yiming Qian and Yasutaka Furukawa. Learning Pairwise Inter-Plane Relations for Piecewise Planar Reconstruction. In *ECCV*, 2020. (Cited on page 22.)

[QYSG17] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NIPS*, 2017. (Cited on page 22.)

[QZN14] Rongqi Qiu, Qian-Yi Zhou, and Ulrich Neumann. Pipe-run extraction and reconstruction from point clouds. In *ECCV*, 2014. (Cited on page 20.)

[RAB18] Carolina Raposo, Michel Antunes, and Joao P. Barreto. Piecewise-Planar StereoScan: Sequential Structure and Motion Using Plane Primitives. *TPAMI*, 40(8), 2018. (Cited on page 55.)

[RCP+13] Rahul Raguram, Ondrej Chum, Marc Pollefeys, Jiri Matas, and Jan-Michael Frahm. USAC: A Universal Framework for Random Sample Consensus. *TPAMI*, 35(8), 2013. (Cited on page 20.)

[RMSG21] Zhongzheng Ren, Ishan Misra, Alexander G. Schwing, and Rohit Girdhar. 3D Spatial Recognition Without Spatially Labeled 3D. In *CVPR*, 2021. (Cited on page 55.)

[RRQ+21] Chiara Romanengo, Andrea Raffo, Yifan Qie, Nabil Anwer, and Bianca Falcidieno. Fit4CAD: A Point Cloud Benchmark for Fitting Simple Geometric Primitives in CAD Objects. In *3D Object Retrieval workshop*, 2021. (Cited on page 22.)

[RVDHV06] Tahir Rabbani, Frank Van Den Heuvel, and George Vosselmann. Segmentation of point clouds using smoothness constraint. *International archives of photogrammetry, remote sensing and spatial information sciences*, 36(5), 2006. (Cited on pages 20, 56, 61, 65, 66, 68, 91 and 93.)

[SB20] Daniel Sieger and Mario Botsch. The Polygon Mesh Processing Library, 2020. (Cited on pages 45 and 48.)

[SDK09] Ruwen Schnabel, Patrick Degener, and Reinhard Klein. Completion and reconstruction with primitive shapes. In *Computer Graphics Forum*, volume 28, 2009. (Cited on page 24.)

[SH20] Gerhard Schrotter and Christian Hürzeler. The digital twin of the city of Zurich for urban planning. *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 88(1), 2020. (Cited on page 1.)

[SHY21] Ehab Shahat, Chang T Hyun, and Chunho Yeom. City digital twin potentials: A review and research agenda. *Sustainability*, 13(6), 2021. (Cited on page 1.)

[SLA15] David Salinas, Florent Lafarge, and Pierre Alliez. Structure-Aware Mesh Decimation. *Computer Graphics Forum*, 34(6), 2015. (Cited on pages 18 and 72.)

[SLK+20] Gopal Sharma, Difan Liu, Evangelos Kalogerakis, Subhransu Maji, Siddhartha Chaudhuri, and Radomír Měch. ParSeNet: A Parametric Surface Fitting Network for 3D Point Clouds, 2020. (Cited on pages 21, 56, 65, 66, 67 and 68.)

[SM19] Bo Sun and Philippos Mordohai. Oriented Point Sampling for Plane Detection in Unorganized Point Clouds. In *ICRA*, 2019. (Cited on page 20.)

[SRF+14] Julian Straub, Guy Rosman, Oren Freifeld, John Leonard, and John Fisher. A Mixture of Manhattan Frames: Beyond the Manhattan World. In *CVPR*, 2014. (Cited on pages 22 and 87.)

[SSBC20] Christiane Sommer, Yumin Sun, Erik Bylow, and Daniel Cremers. PrimiTect: Fast Continuous Hough Voting for Primitive Detection. In *ICRA*, 2020. (Cited on page 20.)

[Str61] Dirk Jan Struik. *"Lectures on Classical Differential Geometry"*. Courier Corporation, 1961. (Cited on page 69.)

[SWK07] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient RANSAC for point-cloud shape detection. *Computer graphics forum*, 26(2), 2007. (Cited on pages 20, 56, 61, 65, 66 and 68.)

[SZP20] Martin Skrodzki, Eric Zimmermann, and Konrad Polthier. Variational Shape Approximation of Point Set Surfaces. *Computer Aided Geometric Design*, 80, 2020. (Cited on page 21.)

[TF22] Hans Tangelder and Andreas Fabri. dD Spatial Searching. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.3 edition, 2022. (Cited on page 98.)

[The18] The Hague Municipality. 3D Stadsmodel Den Haag 2018 CityGML, 2018. https://ckan.dataplatform.nl/dataset/36049d1a-4a0f-4c5d-8adb-21dbfb7252f9/resource/8fb03a5a-5872-4bd6-afd6-9093c1e2e87a/download/39_binckhorst.zip. (Cited on page 44.)

[TK20] Philip Trettner and Leif Kobbelt. Fast and Robust QEF Minimization using Probabilistic Quadrics. In *Computer Graphics Forum*, volume 39, 2020. (Cited on page 18.)

[VKVLV11] Marc Van Kreveld, Thijs Van Lankveld, and Remco C Veltkamp. On the shape of a set of points and lines in the plane. In *Computer Graphics Forum*, volume 30, 2011. (Cited on page 23.)

[VKVLV13] Marc Van Kreveld, Thijs Van Lankveld, and Remco C Veltkamp. Watertight scenes from urban lidar and planar surfaces. In *Computer Graphics Forum*, volume 32, 2013. (Cited on page 24.)

[VSS14] Rebekka Volk, Julian Stengel, and Frank Schultmann. Building Information Modeling (BIM) for existing buildings—Literature review and future needs. *Automation in construction*, 38, 2014. (Cited on page 1.)

[VTHLB15] Anh-Vu Vo, Linh Truong-Hong, Debra F Laefer, and Michela Bertolotto. Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104, 2015. (Cited on page 20.)

[WK05] Jianhua Wu and Leif Kobbelt. Structure Recovery via Hybrid Variational Surface Approximation. *Computer Graphics Forum*, 24(3), 2005. (Cited on page 21.)

[WLL+12] Xiaochao Wang, Xiuping Liu, Linfa Lu, Baojun Li, Junjie Cao, Baocai Yin, and Xiquan Shi. Automatic hole-filling of CAD models with feature-preserving. *Computers & Graphics*, 36(2), 2012. (Cited on pages 13 and 14.)

[YLL+18] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. MVSNet: Depth inference for unstructured multi-view stereo. In *ECCV*, 2018. (Cited on page 9.)

[YNK+18] Zhihang Yao, Claus Nagel, Felix Kunde, György Hudra, Philipp Willkomm, Andreas Donaubauer, Thomas Adolphi, and Thomas H Kolbe. 3DCityDB-a 3D geodatabase solution

for the management, analysis, and visualization of semantic 3D city models based on CityGML. *Open Geospatial Data, Software and Standards*, 3(1), 2018. (Cited on page 2.)

[YYM+21] Siming Yan, Zhenpei Yang, Chongyang Ma, Haibin Huang, Etienne Vouga, and Qixing Huang. HPNet: Deep Primitive Segmentation Using Hybrid Representations. In *ICCV*, 2021. (Cited on pages 22, 65, 66, 67 and 68.)

[YZ18] Fengting Yang and Zihan Zhou. Recovering 3D planes from a single image via convolutional neural networks. In *ECCV*, 2018. (Cited on page 22.)

[ZAB+21] Tong Zhao, Pierre Alliez, Tamy Boubekeur, Laurent Busé, and Jean-Marc Thiery. Progressive Discrete Domains for Implicit Surface Reconstruction. *Computer Graphics Forum*, 40(5), 2021. (Cited on page 17.)

[ZJ16] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10, 000 3d-printing models. *arXiv:1605.04797*, 2016. (Cited on pages 8 and 22.)

[ZJM12] Zihan Zhou, Hailin Jin, and Yi Ma. Robust plane-based structure from motion. In *CVPR*, 2012. (Cited on page 55.)

[ZLSF18] Junqiao Zhao, Hugo Ledoux, Jantien Stoter, and Tiantian Feng. HSW: Heuristic Shrink-wrapping for automatically repairing solid-based CityGML LOD2 building models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 146, 2018. (Cited on pages 6, 16, 44 and 47.)

[ZSGH18] Lingjie Zhu, Shuhan Shen, Xiang Gao, and Zhanyi Hu. Large Scale Urban Scene Modeling from MVS Meshes. In *ECCV*, 2018. (Cited on page 55.)

[ZSL14] Junqiao Zhao, Jantien Stoter, and Hugo Ledoux. A framework for the automatic geometric repair of CityGML models. In *Cartography from pole to pole*. 2014. (Cited on page 16.)

[ZYY+17] Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3D-PRNN: Generating Shape Primitives with Recurrent Neural Networks. In *ICCV*, 2017. (Cited on page 22.)

[ZZX+16] Chen Zhu, Zihan Zhou, Zirang Xing, Yanbing Dong, Yi Ma, and Jingyi Yu. Robust Plane-based Calibration of Multiple Non-Overlapping Cameras. In *3DV*, 2016. (Cited on page 55.)