# 2D Arrangements for Public Space Mapping and Transportation

Murat Yirci

UNIVERSITÉ PARIS-EST

École Doctorale MSTIC
Sciences et Technologies de l'Information Géographique

Murat YIRCI

# Arrangements 2D pour la Cartographie de l'Espace Public et des Transports

Présentée pour obtenir le grade de docteur
de Université PARIS-EST
Avril, 2016

Jury de Thèse

| | |
|---|---|
| Hégron GÉRARD | Président |
| Monika SESTER | Rapporteurs |
| Guillaume DAMIAND | |
| Raouf BEN JEMAA | Examinateur |
| Nicolas PAPARODITIS | Directeur |
| Mathieu BRÉDIF | Encadrant |

This thesis has been conducted at the MATIS (Méthodes d'Analyses pour le Traitement d'Images et la Stéréorestitution) Laboratory of the Research Department of IGN (Institut National de l'Information Géographique et Forestière).

Cette thèse s'est déroulée au laboratoire MATIS (Méthodes d'Analyses pour le Traitement d'Images et la Stéréorestitution) du Service de la Recherche de l'Institut National de l'Information Géographique et Forestière (IGN).

Laboratoire MATIS
Service de la Recherche
Institut National de l'Information Géographique et Forestière
73 avenue de Paris
94165 Saint-Mandé cedex FRANCE

Téléphone : 01 43 98 80 00

# Abstract

## 2D Arrangements for Public Space Mapping and Transportation

This thesis addresses easy and effective development of mapping and transportation applications which especially focuses on the generation of pedestrian networks for applications like navigation, itinerary calculation, accessibility analysis and urban planning. In order to achieve this goal, we proposed a two layered data model which encodes the public space into a hierarchy of semantic geospatial objects. At the lower level, the 2D geometry of the geospatial objects are captured using a planar partition which is represented as a topological 2D arrangement. This representation of a planar partition allows efficient and effective geometry processing and easy maintenance and validation throughout the editions when the geometry or topology of an object is modified. At the upper layer, the semantic and thematic aspects of geospatial objects are modelled and managed. The hierarchy between these objects is maintained using a directed acyclic graph (DAG) in which the leaf nodes correspond to the geometric primitives of the 2D arrangement and the higher level nodes represent the aggregated semantic geospatial objects at different levels. We integrated the proposed data model into our GIS framework called StreetMaker together with a set of generic algorithms and basic GIS capabilities. This framework is then rich enough to generate pedestrian network graphs automatically. In fact, within an accessibility analysis project, the full proposed pipeline was successfully used on two sites to produce pedestrian network graphs from various types of input data: existing GIS vector maps, semi-automatically created vector data and vector objects extracted from Mobile Mapping lidar point clouds.

While modelling 2D ground surfaces may be sufficient for 2D GIS applications, 3D GIS applications require 3D models of the environment. 3D modelling is a very broad topic but as a first step to such 3D models, we focused on the semi-automatic modelling of geospatial objects (such as poles, lampposts, tree trunks, etc.) which can be modelled or approximated by generalized cylinders from single images. The developed methods and techniques are presented and discussed.

**Keywords :** *2D Arrangements, Alpha shapes, Centreline generation, Computational geometry, Delaunay triangulations, Directed Acyclic Graphs (DAG), Geographic Information Systems (GIS), Medial axis, Hierarchical GIS modelling, Pedestrian network graphs, Planar partition, Robust exact geometric computation, Straight skeleton, StreetMaker, 3D generalized cylinder modelling*

# Résumé

## Arrangements 2D pour la Cartographie de l'Espace Public et des Transports

Cette thèse porte sur le développement facilité d'applications de cartographie et de transport, plus particulièrement sur la génération de réseaux piétonniers pour des applications telles que la navigation, le calcul d'itinéraires, l'analyse d'accessibilité et l'urbanisme. Afin d'atteindre ce but, nous proposons un modèle de données à deux couches qui cartographie l'espace public dans une hiérarchie d'objets géospatiaux sémantisés. A bas niveau, la géométrie 2D des objets géospatiaux est représentée par une partition planaire, modélisée par une structure topologique d'arrangement 2D. Cette représentation permet des traitements géométriques efficaces et efficients, ainsi qu'une maintenance et une validation aisée au fur et à mesure des éditions lorsque la géométrie ou la topologie d'un objet sont modifiées. A haut niveau, les aspects sémantiques et thématiques des objets géospatiaux sont modélisés et gérés. La hiérarchie entre ces objets est maintenue à travers un graphe dirigé acyclique dans lequel les feuilles correspondent à des primitives géométriques de l'arrangement 2D et les noeuds de plus haut niveau représentent les objets géospatiaux sémantiques plus ou moins aggrégés. Nous avons intégré le modèle de données proposé dans un framework SIG nommé StreetMaker en complément d'un ensemble d'algorithmes génériques et de capacités SIG basiques. Ce framework est alors assez riche pour générer automatiquement des graphes de réseau piétonnier. En effet, dans le cadre d'un projet d'analyse d'accessibilité, le flux de traitement proposé a permis de produire avec succès sur deux sites un graphe de réseau piétonnier à partir de données en entrées variées : des cartes vectorielles existantes, des données vectorielles créées semi-automatiquement et des objets vectoriels extraits d'un nuage de points lidar issu d'une acquisition de cartographie mobile.

Alors que la modélisation 2D de la surface du sol est suffisante pour les applications SIG 2D, les applications SIG 3D nécessitent des modèles 3D de l'environnement. La modélisation 3D est un sujet très large mais, dans un premier pas vers cette modélisation 3D, nous nous sommes concentrés sur la modélisation semi-automatique d'objets de type cylindre généralisé (tels que les poteaux, les lampadaires, les troncs d'arbre, etc) à partir d'une seule image. Les méthodes et techniques développées sont présentées et discutées.

**Mots Clés :** *Arrangements 2D, Génération de la ligne centrale, Géométrie algorithmique, Triangulation de Delaunay, Graphes dirigés acycliques, Systèmes d'information géographique (SIG), Sciences de l'Information Géographique, axe médian, Modélisation hiérarchique SIG, Graphe de navigation piétionne, partition planaire, calcul géométrique exact et robuste, Squelette droit, StreetMaker, Modélisation de cylindres 3D généralisés*

*"To love and be loved is to feel the sun from both sides."*

*David Viscott*

*to my wife & my parents*

# Acknowledgements

This thesis has been conducted in IGN at MATIS Laboratory with the collaboration of Université PARIS-EST, École Doctorale MSTIC. Thanks to many other people that I have not met in person to express my thanks for making this PhD thesis possible.

I would like to thank my supervisors Nicolas Pararoditis and Mathieu Brédif. Nicolas has been a great director with his continuous support, patience and immense knowledge in his guidance. Mathieu has always been close to me. I have met only a very few people being so nice and smart at the same time. He has been a great mentor in doing my research, sharing his extremely valuable ideas and backing me up.

A perfect place would not be perfect without the perfect people filling the perfect place. That's how I would like to describe the environment in MATIS with full of kind, warm and bright people. As a foreigner, I never felt alone. There has always been a lot of friendly people around. Thanks to Adrien, Alexandre D., Alexandre H., Antonio, Arnaud, Athanasios, Bahman, Bertrand, Bénédicte, Bruno, Clément, David C., David V., Emmanuel, Fabien, Fabrice, Isabelle, Jean Pascal, Jean Pierre, Jérôme, Julien, Kamel, Lâmân, Laurent C., Laurent S., Lijun, Lionel, Loïc, Marie Calude, Martin, Mathieu Bernard, Miloud, Narut, Neelanjan, Nicolas D., Quoc, Rémi, Sébastien, Sidonie, Tristan, Valérie, Xiao, Xiaozhi and Yuchu. It is a pleasure to have met you.

Dear Mom and Dad, I cannot thank you enough for all the support and love you have given me throughout my life. I wouldn't be the person I am today if it wasn't for your constant confidence in me. My sisters Filiz, Tülay and Şerife, thank you for being very nice to me and thank you for giving birth to my loveliest nieces Özgü, Ela and Yasemin and nephews Ozan and Taner. I always feel much better when I think of you. And, my brothers in law Sancar and Ibrahim, my parents in law Serap and Halit, and my sisters in law Didem and Gizem, thank you all for being part of such a gorgeous family.

Finally, to my caring, loving, and supportive wife, Çiğdem. I would like to express my deepest gratitude to you for always being beside me and ready to encourage me when the times get rough.

# Contents

# List of Figures

# List of Tables

# Acronyms

**CAD** Computer Aided Design

**CGAL** The Computational Geometry Algorithms Library

**CGC** Circular Generalized Cylinder

**CHGC** Circular Homogeneous Generalized Cylinder

**CORINE** Coordination of Information on the Environment

**CSG** Constructive Solid Geometry

**DAG** Directed Acyclic Graph

**DCEL** Doubly Connected Edge List

**DEM** Digital Elevation Model

**EEA** European Environmental Agency

**ESRI** Environmental Systems Research Institute

**GDAL** Geospatial Data Abstraction Library

**GIS** Geographic Information System

**GIScience** Geographic Information Science

**GMP** GNU Multiple Precision Arithmetic Library

**GPS** Global Positioning System

**GRASS** Geographic Resources Analysis Support System

**GUI** Graphical User Interface

**HGC** Homogeneous Generalized Cylinder

**IGN** Institut National de l'Information Geographique et Forestière

**ISVD** International Symposium on Voronoi Diagrams in Science and Engineering

**JTS** Java Topology Suite

**LCT** Lower Common Tangent

**MAT** Medial Axis Transformation

**MLS** Mobile Laser Scanning

**MMS** Mobile Mapping System

**NURBS** Non-Uniform Rational Basis Spline

**OFF** Object File Format

**OGC** Oblique Generalized Cylinder

**OGC** Open GIS Consortium

**PGC** Polygonal Generalized Cylinder

**PPVL** Planar Partition Vector Layer

**PRM** Probabilistic Roadmap Method

**RGC** Right Generalized Cylinder

**SDG** Segment Delaunay Graph

**SGC** Straight Generalized Cylinder

**SHGC** Straight Homogeneous Generalized Cylinder

**SPPVL** Semantic Planar Partition Vector Layer

**STL** Stereolithography

**SVD** Segment Voronoi Diagram

**TGC** Toroidal Generalized Cylinder

**TIN** Triangulated Irregular Network

**UCT** Upper Common Tangent

**UGC** Uniform Generalized Cylinder

**UUID** Universally Unique Identifier

**VRML** Virtual Reality Modelling Language

**XML** EXtensible Markup Language

# Chapter 1

# Introduction

## 1.1 Context

Geographic Information Science (GIScience) is a combination of many different fields such as cartography, geodesy, geography, computer science, computer vision, computational geometry, statistics and social sciences. Mark [2003] quoted the definition of GIScience from UCGIS[1] as "The development and use of theories, methods, technology and data for understanding geographic processes, relationships and patterns. The transformation of geographic data into useful information is central to GIScience." Furthermore, Goodchild [1990, 1992] gave a list of GIScience related research topics (*Table* 1.1).

| No | Topic |
|---|---|
| 1. | Spatial analysis and spatial statistics |
| 2. | Spatial relationships and database structures |
| 3. | Artificial intelligence and expert systems |
| 4. | Visualization of spatial data |
| 5. | Social, economic and institutional issues |
| 6. | Data capture, collection and measurement |
| 7. | Data modelling and theories of spatial data |
| 8. | Data structures, algorithms and processes |

**Table 1.1:** A list of research topics in GIScience [Goodchild, 1990, 1992].

Huisman and De By [2009] defined Geographic Information System (GIS) as a computer-based system that provides four basic capabilities (*Table* 1.2) to handle geographic data. Although, sometimes the terms GIS and GIS application are often used in the same meaning without causing any ambiguity, the meanings of the two are different. In fact, a GIS is a tool which can be used to develop different GIS applications. For instance, Geographic Resources Analysis Support System (GRASS) GIS is a free open source GIS which can be used to develop many different applications in various fields.[2] In addition, the interested reader may refer to Pierce and Clay [2007] for a set of GIS applications in agriculture using different GISs.

[1] University Consortium for Geographic Information Science, www.ucgis.org

[2] GRASS GIS applications: https://grass.osgeo.org/documentation/applications/

| Num. | Capabilities |
|---|---|
| 1. | Data capture and preparation |
| 2. | Data management, including storage and maintenance |
| 3. | Data manipulation and analysis |
| 4. | Data presentation |

**Table 1.2:** GIS capabilities [Huisman and De By, 2009].

The context of this thesis is related to developing GIS applications especially related to mapping and transportation with one specific application in mind: generating realistic pedestrian network graphs which avoid static obstacles on the walkways. Such an application involves many research topics (*Table* 1.1) and GIS capabilities (*Table* 1.2). In the following sections, we will discuss the topic and aims of the thesis and how it is related to the context outlined here.

We will see that 2D models of the ground surfaces are required for the generation of the desired pedestrian networks. That is, 2D outlines (footprints) of the geospatial objects are required for 2D GIS applications. Similarly, we will conclude that 3D models of the objects are required for 3D GIS applications. As a first step into 3D modelling for 3D GIS applications, a research work has also been carried out. However, 3D modelling is a very broad topic by itself, therefore we restrict our research on the 3D modelling of the generalized cylinders from single-view.

## 1.2 Topic and Aims

In this thesis, we aimed for a pedestrian network generation process which produces geo-referenced, geometrically true and static obstacle avoiding graphs. That is, the desired graph nodes are geo-referenced and the graph edges represent the true centrelines of the walkways. In addition, we wanted to analyse such a GIS application development process and if possible design a framework for developing similar mapping and transportation applications. The requirements of such a GIS framework is discussed along the pedestrian network generation process which are more or less applicable to similar applications with minor modifications.

A typical GIS application has a very basic structure at the very highest level. Geospatial data are processed with a set of algorithms in order to provide the desired outputs. Such a basic structure / schema of a GIS application is given in *Figure* 1.1. In this schema, the data model is a bridge between the algorithms and the data. Algorithms operate on the data through the data model, therefore the design of the data model is strictly related to the type of the data (*Chapter* 2 briefly introduces the data types in GIScience).

The surfaces on which the pedestrians can freely move are called walkways. Walkways are composed of geospatial urban objects such as sidewalks, parks, public squares, pedestrian crossings, etc. The rest of the urban objects (e.g. buildings, city furnitures, motorways, etc.) are considered as barriers for the movement of pedestrians. The footprints of all these objects define a partition on the surface of the cities. Therefore, when the earth

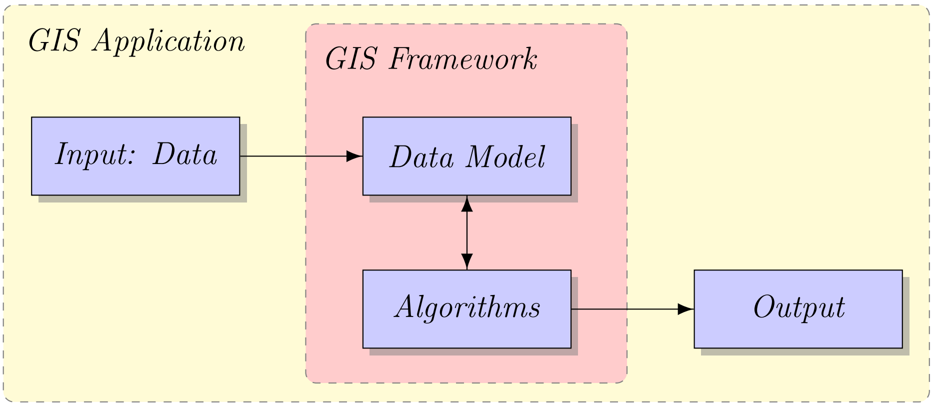


**Figure 1.1:** A typical GIS application schema.

surface is modelled in 2D, planar partitions[3] are the natural choice for describing different regions (i.e. outlines of the geospatial objects) on the surface.

Planar partitions can either be created from scratch or existing (previously created) vector datasets can be leveraged. In both cases, the GIS capabilities given in *Table* 1.2 are required for fixing the geometric and topological errors and/or completing the missing data. Furthermore, upon creation of planar partitions they have to be validated. The theory of valid planar partitions are discussed in *Chapter* 2 but in short, a valid planar partition has to comply with the following three rules: (i) all polygons must be valid, (ii) no gaps between the polygons are allowed and (iii) polygons must not overlap.

In a dynamic system managing planar partitions, the geometry and the topology of the primitives[4] describing the partitions might change occasionally. Then, in order to achieve a dynamic system, planar partitions are able to be modified by adding new geometry or deleting the existing ones. After applying the updates, it is also crucial to keep the planar partitions valid for the integrity of the data model (*Figure* 1.2). The *"Maintain"* block in *Figure* 1.2 relates to the finding of the intersections between the line segments (between the existing and the newly inserted ones), splitting the existing lines and polygons, removing polygons, etc. for keeping the modified planar partitions valid.

Besides the geometry, we also want to capture the thematic and semantic information related to geospatial urban objects (*Chapter* 3). An object oriented approach fits very well for grouping the related information over a planar partition. We create abstract software objects called semantic objects that integrate the aforementioned geometric and semantic information. The data model should also count for the natural hierarchy between the geospatial objects. That is, objects sharing the same physical surface have to be considered. For instance, in a street scene, a top level object can be the street itself composed of the sidewalks, roads and buildings.

Traditionally, planar partitions are represented by a set of polygons which are used to model only areal objects. On the other hand, geospatial objects can also be linear and punctual. Therefore, in order to represent all types of objects within a single framework, the data model based on the planar partitions should be extended.

Maintenance of the underlying geometry and topology (i.e. planar partitions) is only the first part of a bigger maintenance step. As the geometry and the topology change,

[3] A planar partition is the tessellation of the plane into non-overlapping polygons.

[4] Geometric primitives composing planar partitions: points, line segments and polygons.

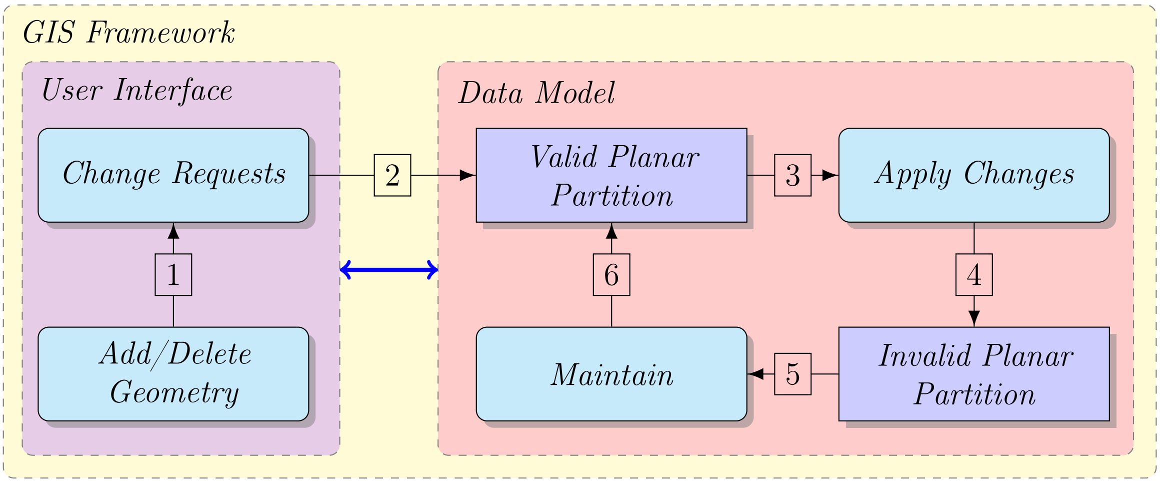


**Figure 1.2:** Maintenance of the planar partitions. The numbers in the figure indicate the order of operations in a conceptual GIS for keeping managed planar partition valid, i.e. keeping the data model consistent.

already constructed objects also need to be maintained (*Section* 3.4). For this reason, each semantic object keeps a list of geometric primitives that it was constructed on. Then, the maintenance of the semantic objects is achieved by keeping these geometry lists up-to-date. The rule here is that once an object has been created, it's initial geometry must not shrink, enlarge, etc. For instance, a newly added line segment can split an existing polygon which had been previously linked to a semantic object. Therefore, the other half of the split polygon has to be added to the geometry list of the corresponding semantic object in order to keep the initial geometry of the object effectively-unchanged[5].

Having the explicit topological relationships between the geometric primitives (e.g. which edges are connected to a given point, which edges circulate around a given polygon, which polygons are neighbours to each other, etc.) is important for efficient and effective geometric processing (*Section* 2.2). For instance, the maintenance issues introduced previously can benefit a lot from a data model which stores explicit topological information.

Computational geometry and scientific computing algorithms need to be robust (*Section* 5.5). Storing real numbers with fixed-length bits within computers using the floating point format defined by ANSI/IEEE Standard 754 [1985] may cause the programs to crash or produce erroneous outputs. Therefore, GIS applications performing geometric computations need to have robust data models and algorithms in order to prevent these errors.

*Tables* 1.3 and 1.4 summarize the requirements of a data model and a GIS framework that can be used to accomplish the desired pedestrian network generation application. According to the identified requirements, we proposed to represent planar partitions with 2D arrangements of line segments which subdivide the plane into 0-dim points (vertices), 1-dim line segments (edges) and 2-dim polygons (faces). Besides planar partitions, 2D arrangements can also support isolated points[6] and individual line segments[7] on which punctual and linear semantic objects can be constructed respectively. In fact, we will

[5] The geometry of the object changes but the union of the primitives constructing the object geometry does not change.

[6] Not connected to any line segment.

[7] Not on the boundary of a polygon.

see that 2D arrangements are nothing but generalized planar graphs from which planar partitions can easily be extracted (*Section* 2.4). The benefit of 2D arrangements is that, when they are implemented with a topological data structure, it is very efficient to maintain and validate the underlying planar partitions especially in dynamic environments. In addition, our data model captures the hierarchy between the objects using a Directed Acyclic Graph (DAG) structure (*Section* 3.3). The leaf nodes in this hierarchy do not correspond to objects but geometric primitives. Semantic objects constructed on these primitives correspond to higher level nodes in the hierarchy graph. We will see that such a DAG is also very useful for rendering/displaying the semantic objects on the computer screen (*Section* 3.5). Note that in an object hierarchy sharing the same set of primitives, all objects may not be rendered simultaneously due to the possible shared surfaces and occlusions. The DAG hierarchy is used to find out and resolve occlusions and ambiguity conditions related to object rendering.

| No | Data Model Requirements |
|---|---|
| $R_1$ | It should be able to represent planar partitions. |
| $R_2$ | It should be able to validate planar partitions. |
| $R_3$ | It should be able to maintain the validity of the planar partitions upon adding/deleting geometry. |
| $R_4$ | It should be able to construct software objects that model the geospatial objects possibly with semantic information. |
| $R_5$ | It should be able to represent areal, linear and punctual objects. |
| $R_6$ | It should be able to construct the hierarchy of the objects. |
| $R_7$ | It should be able to keep the already created software objects consistent when the underlying geometry is changed. |
| $R_8$ | It should keep explicit topological information in order to provide efficient and effective geometric computing. |

**Table 1.3:** Data model requirements for supporting mapping and transportation applications.

| No | GIS Framework Requirements |
|---|---|
| $R_1$ | It should have a data model that complies with the requirements given in *Table* 1.3 |
| $R_2$ | It should allow creation of planar partitions from scratch. |
| $R_3$ | It should be able to leverage existing datasets for generating planar partitions. |
| $R_4$ | It should have the basic GIS capabilities for handling requirements $R_2$ and $R_3$. |
| $R_5$ | It should have a set of robust algorithms for geometric computing. |

**Table 1.4:** The GIS framework requirements for supporting mapping and transportation applications.

*Figure* 1.3 displays the overall proposed data model for a synthetic street scene. In the figure, the main components of the data model are displayed in two separate layers. The data model establishes a basis for the GIS applications by capturing the geometry and semantics of the region of interest via planar partitions and semantic objects. Once the data model is up and ready, the application specific algorithms can operate on it.

The proposed data model is encapsulated into our GIS framework called StreetMaker (*Figure* 1.4, refer to *Chapter* 5 for the details). StreetMaker has also a set of basic GIS capabilities for inserting, deleting, selecting, editing and displaying the geometry and creating the semantic objects together with the associated hierarchy. In addition, a few generic algorithms ready to operate on the data model are integrated into StreetMaker. These algorithms can be utilized by any GIS application based on StreetMaker as our pedestrian network generation application. The list of the integrated algorithms and their possible usages are summarized in the *Table* 1.5. The theory and applications of these algorithms are discussed in *Chapter* 4.

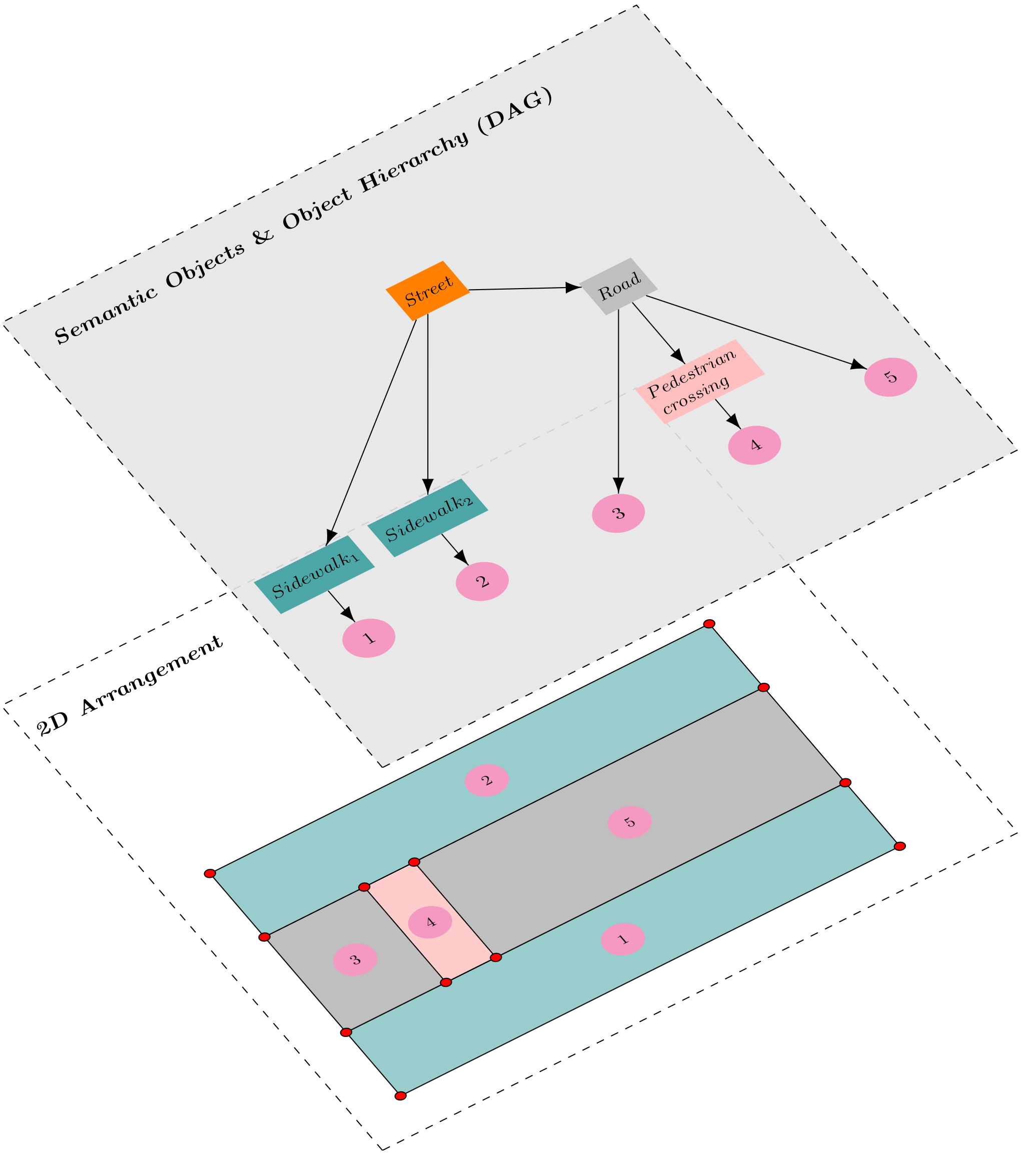
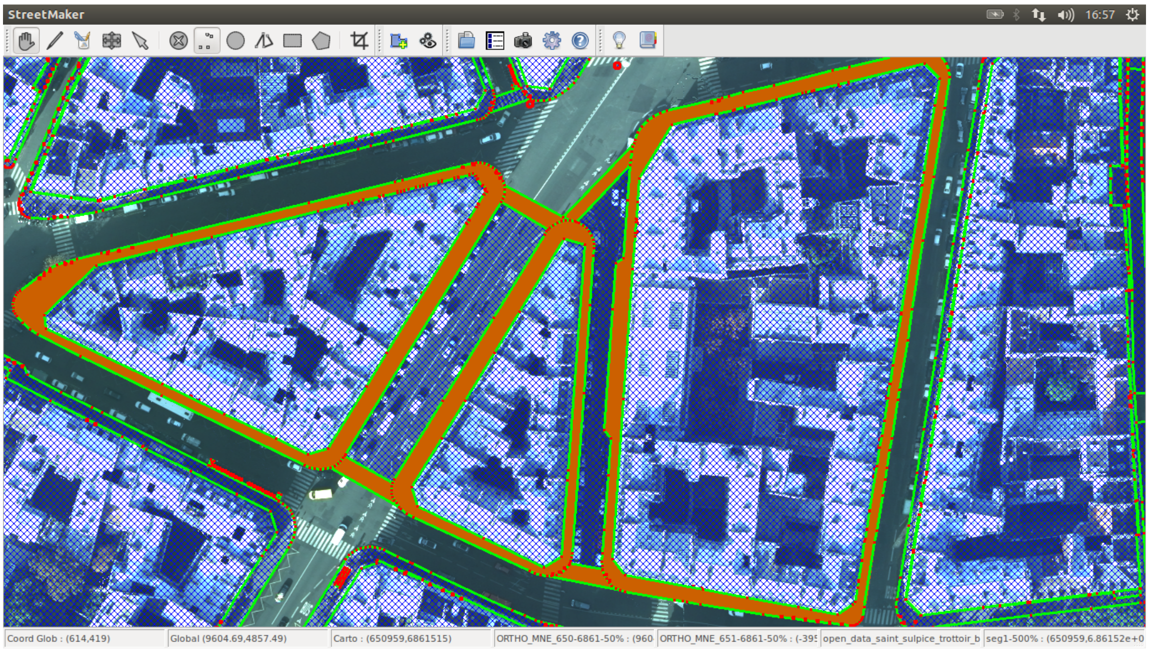


**Figure 1.3:** A street scene is modelled with the proposed data model. At the lowest level, geometric primitives (points, line segments and polygons) constituting the 2D arrangements are used to model the planar partitions. Then, the semantic objects are created on the underlying arrangement cells. The hierarchy between the semantic objects and the geometric primitives are constructed using a DAG.

The proposed pedestrian network generation algorithm is composed of a number of building blocks as displayed in *Figure* 1.5. The algorithm starts with the extraction of the walkways on which the pedestrians can walk (1). Then, the geometric centrelines of the



Coord Glob : (614,419) | Global (9604.69,4857.49) | Carto : (650959,6861515) | ORTHO_MNE_650-6861-50% : (960 | ORTHO_MNE_651-6861-50% : (-395 | open_data_saint_sulpice_trottoir_b | seg1-500% : (650959,6.86152e+0

**Figure 1.4:** StreetMaker: a GIS framework for developing GIS applications. In the figure, semantic objects constructed on the underlying planar partitions are displayed in the main window of StreetMaker. Planar partition and the semantic objects are overlaid on an orthophoto.

| **Algorithm** | **Possible Usages** |
|---|---|
| Delaunay triangulation | Triangulated Irregular Networks (TIN), 2D $\alpha$-shapes, Segment Voronoi Diagrams (SVD) from Segment Delaunay Graphs (SDG). |
| Voronoi diagram | Nearest neighbour, medial axis from SVD, centrelines from medial axis. |
| 2D $\alpha$-shapes | Estimating 2D shapes form 2D point clouds. |
| Straight skeleton | Centrelines, polygon offsets, plausible roof structures, terrain generation. |
| Connectivity graph | Thematic graphs for itinerary calculations. |

**Table 1.5:** Integrated algorithms into StreetMaker and their possible usages in GIS applications

walkways are computed using the medial axis transform (2). These centrelines correspond to the pure pedestrian network graphs (initial graph) which are augmented by inserting additional nodes and structured by identifying groups of connected edges called trajectory arcs (*Figure* 1.6) (3). The additional nodes are computed by intersecting the borderlines (line segments constituting the borders between different objects) and the existing centrelines in the initial graph. Therefore, there will be a node in the final graph for each borderline separating two objects. In addition, some of the graph nodes are marked as special depending on their properties (will be detailed in *Section* 6.2). These special nodes indicate the start and end points of trajectory arcs which represent the parts of the network graph on which an entity can travel without making any further decisions on the path selection. Furthermore, the total length of a trajectory arc and the minimum width between each trajectory arc and the surrounding object borders are calculated. In the following step, the selected punctual semantic objects are projected onto the computed pedestrian graphs (4). The punctual objects generally represent building doors, bus stops, metro stations, etc. and their projections on the graphs correspond to the junctions from which pedestrians can enter or exit the graphs. At this stage of the algorithm, the basic pedestrian network graph is ready. However, since our data model might possess many useful semantic and thematic information (such as the names of the streets, avenues and regions) related to the surrounding geospatial objects, this information can also be utilized within the graphs (5). The more information embedded into the graphs, the more sophisticated algorithms can run on them and the more they become useful for the entities utilizing these graphs for itinerary calculations and navigation. The final step of the process is the exportation of the graph into an EXtensible Markup Language (XML) file so that it might be used by the third parties (6).

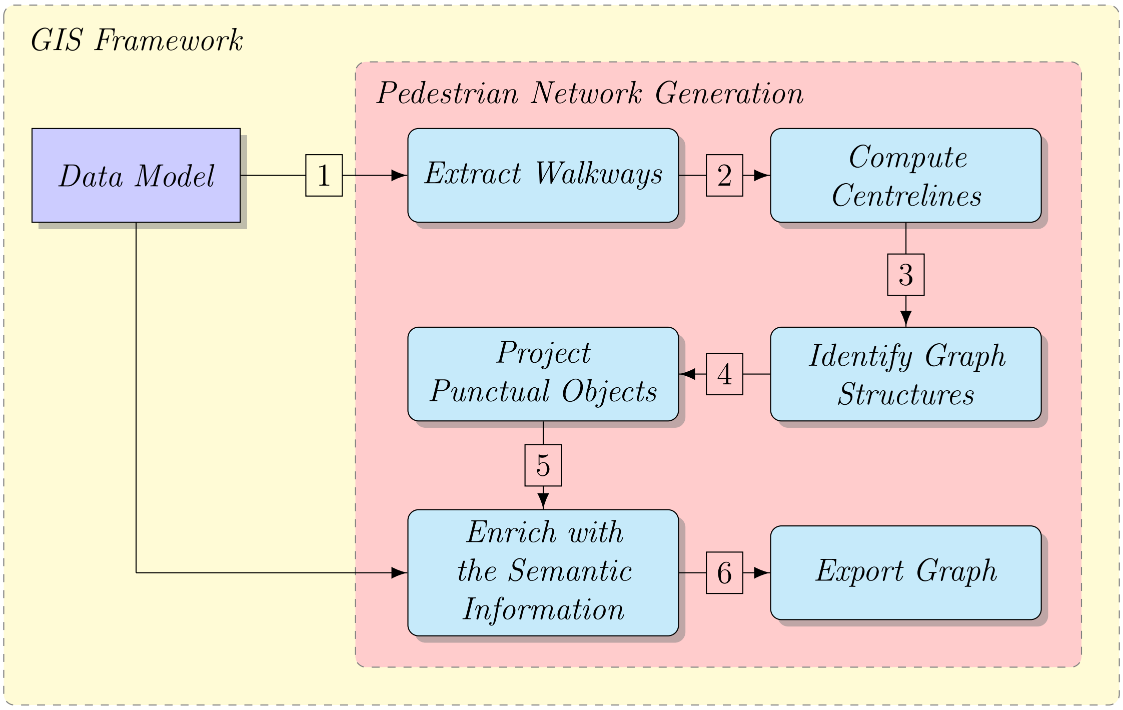


**Figure 1.5:** Building blocks of the pedestrian network generation algorithm. The numbers in the figure indicate the order of execution for the blocks.

More samples of the computed graphs are presented in *Chapter* 6 but in this chapter *Figure* 1.6 is given just for the first glimpse of the computed pedestrian network graphs. The meanings of different symbols used in this figure will be clarified within *Chapter* 6.

The aforementioned term "walkway" is used conceptually within the context of pedestrian network graphs. It could be "roadway" if the graphs were generated for the motorized vehicles. In fact, the same pipeline can be used to generate the network graphs for any other types of travelling entities. It all comes down to the created semantic planar partitions and the filtered semantic objects for graph generation. For instance, sidewalks, pedestrian crossings, city squares, etc., are considered as walkways for the pedestrian network generation and the remaining semantic objects such as buildings, roads and static objects (e.g. city furnitures, poles on the pavements, traffic lamps, etc.) are filtered out since they are classified as barriers to the movement of the pedestrians. On the contrary, for the motorized vehicle network graphs, roads and parking areas can be considered as roadways whereas the rest of the surfaces can be excluded (filtered out) as barriers. In addition, the

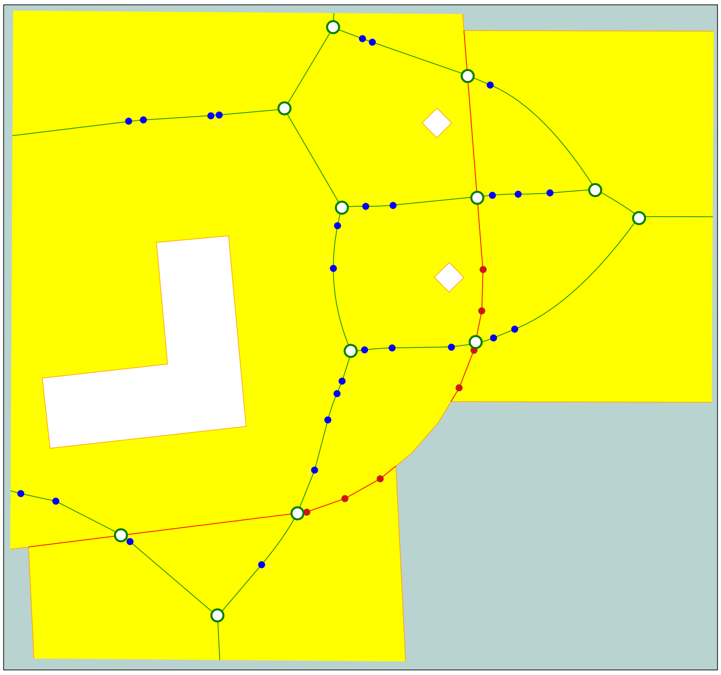


**Figure 1.6:** A part of an example pedestrian network graph at the intersection of a sidewalk and two pedestrian crossings. Colours: walkways (yellow), obstacles (white), centrelines or graph edges (green), graph nodes (blue and white-green), borderlines and border nodes (red).

semantic information captured by the objects can also be used for filtering. An example would be the filtering out of some surfaces (the corresponding semantic objects) which are too steep when generating the graphs for the wheelchair users. As a result, the level of details of the objects as well as the amount of semantic information captured increase the versatility of the proposed algorithm. The more finer the semantic objects and the more semantic information embedded into them, the higher the intelligence for filtering surfaces when generating different types of network graphs.

The proposed pedestrian network generation algorithm is generic meaning that it can be applied to any semantic planar partition. However, creating the semantic planar partitions, i.e. constructing the data model might be different for different applications. In this thesis work, we present two different pipelines for creating two different semantic planar partitions for two different applications. The steps of these pipelines were mostly determined according to the properties of the initial data at hand.

The first application is related to the Saint Sulpice region of Paris. The pipeline used for this application is given in the *Figure* 1.7. In this section, only the brief summary of the pipeline steps is described without going into details. For the detailed descriptions please refer to *Section* 6.4. We utilized the OpenData Paris dataset[8] for generating the base planar partition and Institut National de l'Information Geographique et Forestière (IGN)'s BD Adresse®[9] for the punctual objects corresponding to the building doors. The utilized vector data were converted to the 2D arrangements representation (1 in the *Figure* 1.7). Then, the existing geometric and topological data were first corrected by threshold based automatic error correction tools (2 in the *Figure* 1.7). Note that the threshold values are data dependent and in this application they were decided according to the

[8] http://opendata.paris.fr/page/home/

[9] http://professionnels.ign.fr/bdadresse

characteristics of the input data by doing simple preliminary trial and error tests. The remaining errors were fixed manually in addition to the manual completion of the missing data by overlaying the 2D arrangement on the background geo-referenced orthophotos (3 in the *Figure* 1.7). Once the planar partition was obtained, semantic objects were manually created by grouping the planar partition primitives (4 in the *Figure* 1.7). As the semantic objects were being created, the DAG-structured object hierarchy was also constructed (5 in the *Figure* 1.7). The final step is the running of the pedestrian network generation algorithm on the constructed data model (6 in the *Figure* 1.7).

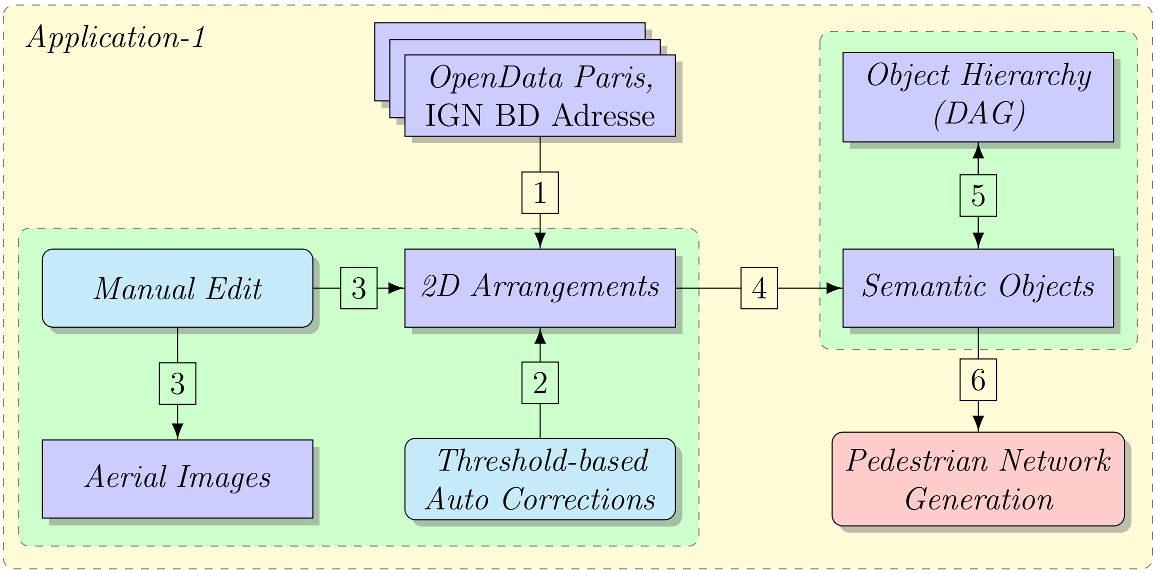


**Figure 1.7:** The pipeline for the first application: Pedestrian network generation for Paris Saint-Sulpice region. The numbers in the figure indicate the order of execution for the blocks.

The region of interest for the second application was the urban conurbation of Saint-Quentin-en-Yvelines (CASQY). The main steps of this pipeline is displayed in the *Figure* 1.8 which are detailed in *Section* 6.5.

For the second application, we followed a different approach in creating the initial semantic planar partition (planar partition + semantic objects). That is, externally created vector data are automatically imported into StreetMaker as semantic objects. Therefore, no manual intervention was required for creating the semantic objects. This kind of approach is possible when the input vector data are well-structured, i.e. they are close to planar partitions if they are not already (*Section* 5.4.3).

Four different data sources were used for creating the semantic planar partition for the second application: CASQY dataset[10] (i), manually created pedestrian crossings (ii), obstacles obtained from a 2D point cloud (iii) and IGN's BD Adresse® (iv). Before importing any semantic objects into StreetMaker, we applied three mini-pipelines {(1,2,3), (4,5,6) and (7,8,9)} (*Figure* 1.8) in order to prepare the data to be imported.

The main input dataset was the CASQY dataset describing the basic land use of the CASQY region. Although it is not a common practice[11] in GIScience, this dataset is close to a valid planar partition except for a few errors which were fixed with a preprocessing

[10] Supplied from Saint-Quentin-en-Yvelines city hall for the CASQY region

[11] In GIScience, vector data are generally created/edited with non-topological GIS which deviates vector data from being planar partitions.

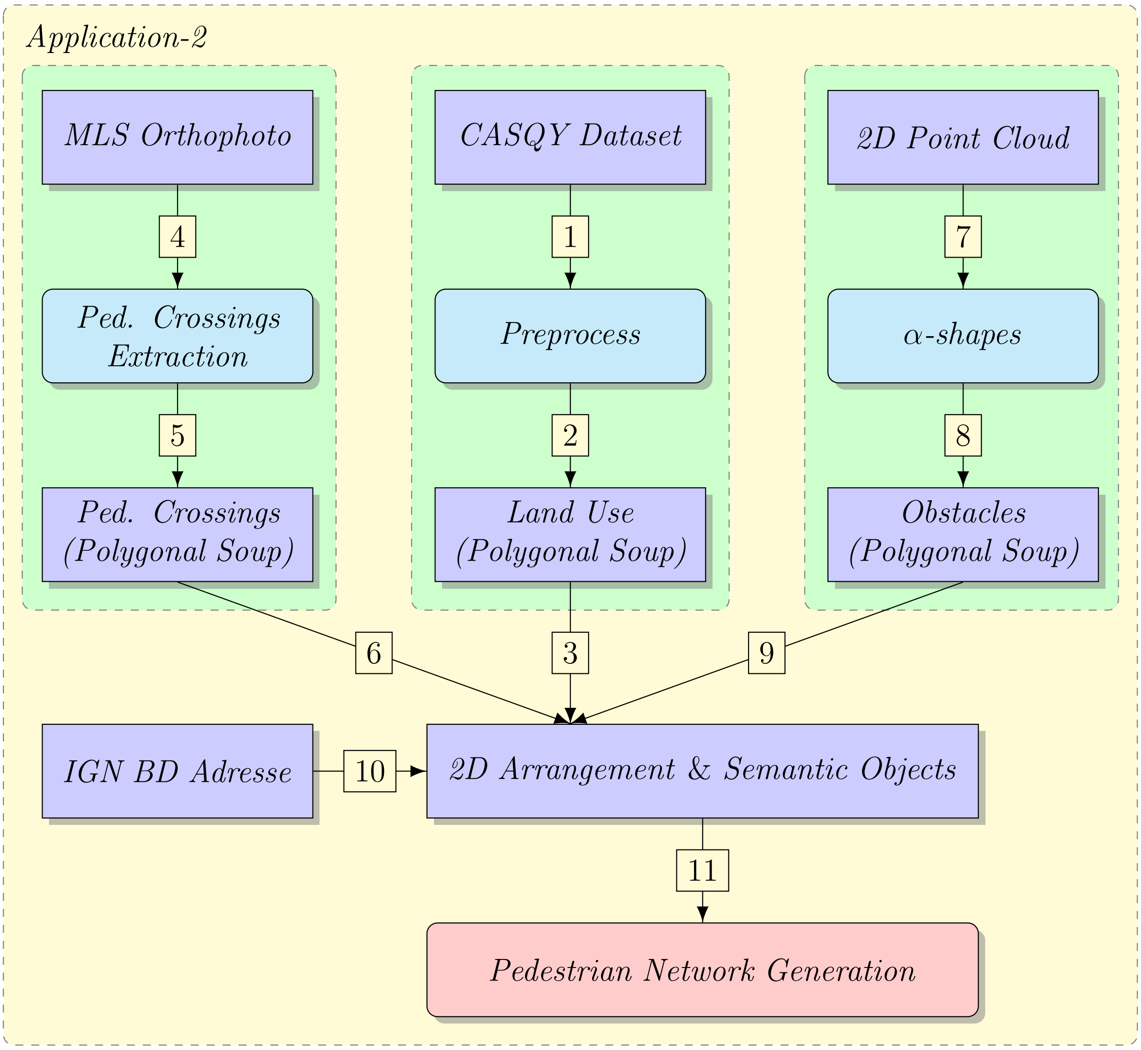

Application-2
MLS Orthophoto
4
Ped. Crossings Extraction
5
Ped. Crossings (Polygonal Soup)
CASQY Dataset
1
Preprocess
2
Land Use (Polygonal Soup)
2D Point Cloud
7
α-shapes
8
Obstacles (Polygonal Soup)
6
3
9
IGN BD Adresse
10
2D Arrangement & Semantic Objects
11
Pedestrian Network Generation

**Figure 1.8:** The pipeline for the second application: Pedestrian network generation for Saint-Quentin-en-Yvelines urban conurbation (CASQY). The numbers in the figure indicate the order of execution for the blocks. In the figure MLS stands for Mobile Laser Scanning and Ped. is the short form of Pedestrian.

step (1 and 2 in the *Figure* 1.8). The CASQY dataset is not a finely detailed land use dataset, it lacks the pedestrian crossings and static obstacles which are crucial for our pedestrian network generation application. Outlines of the pedestrian crossings in the region of interest were captured manually on top of a background Mobile Laser Scanning (MLS) orthophoto (4 and 5 in the *Figure* 1.8) [Vallet and Papelard, 2015; Brédif et al., 2015]. Similarly, outlines of the static obstacles on the surface of the walkways were generated using the $\alpha$-shapes algorithm on a 2D point cloud (7 and 8 in the *Figure* 1.8). The point clouds used for generating both the MLS orthophoto and the 2D point cloud were collected using the IGN's 3D Mobile Mapping System (MMS) called STEREOPOLIS [Paparoditis et al., 2012]. The 2D point cloud was obtained by vertically projecting the 3D points which were classified as parts of static obstacles [Serna and Marcotegui, 2014].

After running the three mini-pipelines and importing the resulted polygonal soups (*Section* 2.2) into StreetMaker together with the address points, the input vector data were converted into a semantic planar partition (3, 6, 9 and 10 in the *Figure* 1.8) on which the pedestrian network generation algorithm was run (11 in the *Figure* 1.8).

The main focus of this thesis work was devoted to the development of a generic data model that can be used to generate static obstacle avoiding pedestrian networks. Once this goal was achieved, an additional research has been done on the development of the 3D GIS applications. As we have already discussed, the 2D models of the environment are needed for the generation of pedestrian networks. Similarly, for many 3D GIS applications, the environments should be modelled in 3D.

A 3D environment is composed of a 2D ground surface embedded in 3D and the 3D objects are positioned on the ground surface. Generating realistic 3D models of the environment (e.g. 3D virtual city models) is a very broad and challenging task which involves many research fields such as computer vision, computer graphics, remote sensing, etc. Considering the large number of different object types (e.g. ground surface, buildings, trees, city furnitures, traffic lamps and signs etc.) to be modelled and the diverse research topics, we narrowed down our interest to the single-view 3D modelling of the objects that are of type generalized cylinder.

A generalized cylinder is defined by set of planar curves that are positioned around an axis curve (see *Section* 7.2.1 for a more formal definition and taxonomy of the generalized cylinders). Objects of type lampposts, poles, tree trunks, signposts, etc. are all in the form of a generalized cylinder. These objects are among the most frequently encountered objects on the surface of the walkways. Recall that for the static obstacle avoiding pedestrian network generation, we need to construct the outlines of the obstacles on the walkways. In fact, once the objects are modelled, the footprints of these objects (a 3D circle on the ground surface for this case) can be utilized in the pedestrian network generation application as obstacles. This was also another motivation of ours while choosing the research direction among many others. However, objects can only be reconstructed upto a scale from a single-view and the resultant models cannot be localised (and scaled) precisely due to lack of depth information. For correctly scaled geo-referenced models, at least two-view of the objects are needed (assuming the camera parameters are known). On the other hand, once the shapes of the objects are reconstructed (modelling objects upto a scale factor), geo-referencing and scaling can be done afterwards which is left as a future work.

We tried to develop an interactive system similar to the work of Chen et al. [2013]. Our modelling system and theirs are compared at various places along the *Chapter* 7. The main idea is the combination of human perception with the computational power of the computers which leads to a semi-automatic modelling system. In this system, users initiate the modelling by drawing the initial 2D profile (ellipse on the image) of a generalized cylinder on the input image. Then, they are asked to approximate the axis of the generalized cylinder by sweeping the initial profile along the image of the generalized cylinder. The major axis of the initial 2D profile is fit to the image outlines as the user drags the profile. This generates the input data for the 3D reconstruction of the generalized cylinder (*Figure* 1.9).

Using the input data, Chen et al. [2013] first generated a set of 3D circles with the assumption of a ratio preserving camera projection and a planar axis whose supporting plane is parallel to the image plane, then optimize the location, orientation and shape (i.e. a 3D circle can be warped into a 3D ellipse) of the initially constructed 3D circles. This optimization tries to compensate for the initially made false assumptions by minimizing a set of geo-semantic (geometric and semantic) constraints between the separately modelled parts. A part in their system is either a generalized cylinder, cuboid or a sphere. Our

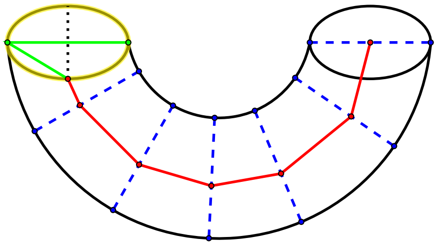


**Figure 1.9:** Input data for 3D generalized cylinder reconstruction: initial 2D elliptic profile (yellow) is generated by drawing two line segments (green), the axis of the generalized cylinder is approximated (red) from the user sweep and the major axis of the initial elliptic profile is fit to the outline of the generalized cylinder in the image at the sampled axis points (blue).

modelling system only supports modelling of generalized cylinders and upgrading it to a part-based modelling system that merges separately modelled parts is left as a future work. On the other hand, our system reconstructs the 3D circles using the perspective projection with a pin-hole camera model. Reconstruction of a 3D circle from its projection is explained in detail including the special and degenerate cases in *Section* 7.2.4.

Notice in *Figure* 1.9 that, only the projection of the first and last 3D circles can be fully captured. For the 3D circles in between, only the major axis of their projections can be estimated. We derived a parametric closed form solution for the position of the 3D circles when the normal (orientation) and the partial projection (only the major axis of the projected ellipse) of the circles are known. Based on this solution, generalized cylinders are modelled using three different prior constraints: constant depth (3D circle centres are assumed to lie on the same plane which is parallel to the image plane), planar axis (3D circle centres are assumed to lie on the same plane) and linear axis (3D circle centres lie on a 3D line).

*Figure* 1.10 displays a generalized cylinder modelled in our system. In *Chpater* 7, more examples are presented along with more detailed discussions.

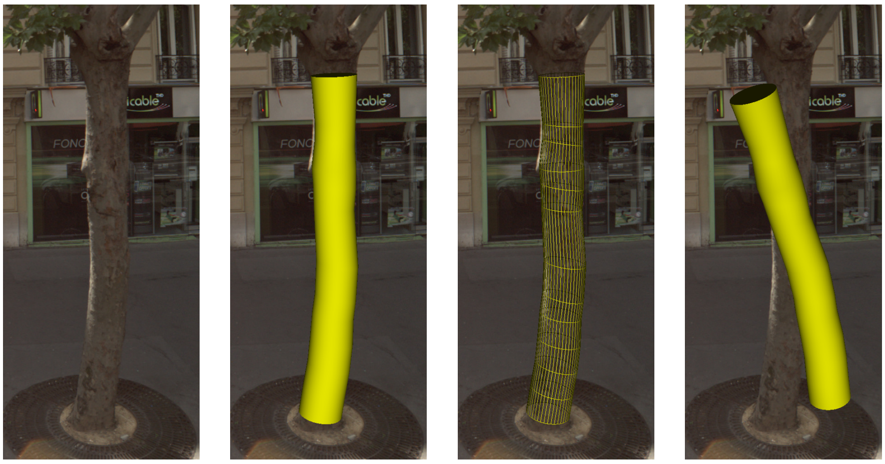


**Figure 1.10:** A modelled tree trunk: input image and the reconstructed model projected on the image space.

## 1.3 Contributions

The contributions of this thesis are three fold.

First, a data model is proposed for representing planar partitions and a hierarchy of geospatial objects. Planar partitions are represented by 2D arrangements and the object hierarchy is captured using a DAG structure. The underlying data structure of the 2D arrangements is a topological data structure which allows efficient and effective geometric computing at the topological level. 2D arrangements representation has the advantage of easy validation and maintenance, especially in dynamic environments in which the data are occasionally updated. Furthermore, we demonstrated that rendering an object hierarchy on the computer screen might be problematic when the parent objects share the same surface through their children. This rendering problem is solved utilizing the DAG hierarchy and to the author's knowledge, this kind of approach is used for the first time. The proposed data model is encapsulated within a GIS framework which also includes a few algorithms integrated for easy GIS application development. This framework is compact, small and well-suited for desktop GIS applications especially for mapping and transportation applications.

Second, a full pipeline for generating static obstacle avoiding pedestrian network graphs is presented with two sample applications. The algorithm takes an input of object hierarchy constructed on top of a planar partition and generates a pedestrian network graph which is geometrically exact. That is, the centrelines of the walkways are geometrically true. The generated graphs are further structured into sub-graphs such that once a sub-graph is entered by a travelling entity it can move freely without giving any further navigational decisions. In addition, for each sub-graph geometrically true lengths and the minimum width to the surrounding objects are computed. With these given properties, the computed graphs are the first examples of such graphs in the literature. The graphs are computed from a set of semantized surfacic objects which also allows embedding the related semantic information directly into the graphs.

Third, computation of 3D circles from their perspective and orthogonal projections are explained in detail including the degenerate and special cases. Furthermore, a parametric closed form solution for the centre of a 3D circle is derived under perspective projection when the normal and the partial projection (only the major axis of the projected ellipse) of the 3D circle are known. Then, this closed formula is used in a novel system for modelling 3D objects of type generalized cylinders by using three different prior constraints.

## 1.4 Structure and Content of the Thesis

The content of the chapters are summarized below.

***Chapter* 1** gives an overview of the content, topic, aims and the claimed contributions of the thesis.

***Chapter* 2** starts with a brief introduction to fundamental data types used in GIS. The chapter continues with the discussion of different vector data representations and it emphasizes the importance of topological data structures for geometric computing. Planar partitions are introduced as a special type of vector data and their theory is discussed using the planar graphs. Furthermore, the issues related to validity and maintenance of planar partitions are raised. Then, as a solution to these issues, representing planar partitions with 2D arrangements is proposed along with detailed discussions.

***Chapter* 3** is related to the object oriented modelling of the geospatial objects on top of the planar partitions. First, the chapter introduces the semantic objects and the natural hierarchy among them. Then, encoding this hierarchy into a DAG is discussed which is later used to solve problems related to semantic object maintenance and rendering.

***Chapter* 4** aims to provide theoretical background and use cases on the integrated generic algorithms that are thought to be useful in many GIS applications.

***Chapter* 5** introduces our GIS framework called StreetMaker for developing GIS applications. The integration of the proposed data model into StreetMaker is explained with the discussion of how semantic planar partitions are created within StreetMaker. Furthermore, our approach to well-known robustness issues in computational geometry is detailed and the chapter is concluded with the example outputs of the integrated generic algorithms that can operate on the created semantic planar partitions.

***Chapter* 6** begins with the state-of-the-art literature for generating pedestrian networks. Then, our pedestrian networks are compared with the previously created ones. Finally, two different application pipelines are introduced which utilize the proposed pedestrian network generation algorithm.

***Chapter* 7** is related to the preliminary work that has been done for 3D GIS application development. The developed system, methods and formulas for modelling 3D generalized cylinders from single images are explained with the necessary discussions.

***Chapter* 8** concludes the manuscript with an overall assessment, future perspectives and some extra ideas on the big picture.

***Appendix* A** lists pseudo codes for some of the geometric data structures. This appendix is complementary to the *Section* 2.2.

***Appendix* B** is devoted to 2D Delaunay triangulations. The first part of the appendix demonstrates a proof for legal and illegal edges of a triangulation. The second part demonstrates the steps of an algorithm that is used to merge two separate Delaunay triangulations into a single one. This appendix is complementary to the *Section* 4.1.

# Chapter 2

# Planar Partions

## 2.1 Introduction

GIScience is a very broad term which includes all the human efforts to capture, visualize, analyse and interpret geospatial data to understand the relationships, find out patterns and trends. As a consequence, a GIS might be very complex involving many technologies, methods and processes in which the data occupies the central role. There has been myriad of data formats used in GISs, however one can classify them under two main groups: raster data and vector data. Very basic introductory paragraphs are given below for both data types; the interested reader can refer to many sources on the internet as well as Lloyd [2010] for more complete definitions and discussions.

Raster data is represented by a grid structure which stores the information within the entries of the grid, called cells. Cells can be considered as pixels of an image. In fact, aerial images are the most commonly used raster data in GISs. A raster can represent both discrete/thematic data (e.g. land use) and continuous data (e.g. temperature and elevation). However, due to its discrete nature, information is lost when continuous data has to be sampled (*Figure* 2.1). The accuracy of the data depends on the resolution of the sampling grid. On the other hand, since it is space oriented, it provides direct access to information about a given location and its regular structure helps organizing spatial information.

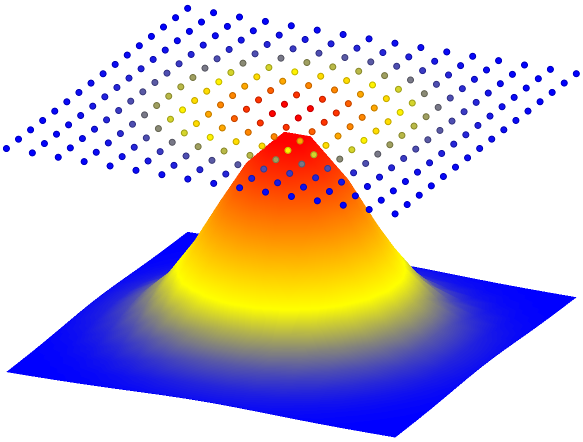


**Figure 2.1:** Raster representation of an elevation data.

A vector in GIScience describes a coordinate-based data model that represents geographic features using geometric primitives: points, lines, and polygons. In 2D, each point is represented by a coordinate pair and other geometric features (e.g. lines and polygons) are represented as ordered lists of points. The main advantage over the raster representation is that vectors can store continuous geospatial data without loss of information. For example, vectors can be magnified without loss of quality when rendered on the computer screen, while the rasters cannot. On the other hand, vector data requires more complex data structures for processing and indexing.

A planar partition is a type of vector data that defines a map on a planar surface by subdividing it into non-overlapping polygonal regions. Therefore, planar partitions are very suitable for representing 2D surfaces on which polygonal regions need to be classified/categorized. Recall from *Chapter* 1 that we needed to model the footprints of the geospatial objects on the ground surfaces of the cities. Consequently, planar partitions fit very well to our requirements with the prospect of mapping a set of subdivided polygonal regions to the footprints of a geospatial object.

This chapter is devoted to the analysis of planar partitions. First, an overview of the existing geometric data structures for vector data representation is presented (*Section* 2.2). Second, the theory of the planar partitions are explained using the graph theory. In addition, the issues for maintaining valid planar partitions in a dynamic environment are discussed along with the data structure requirements (*Section* 2.3). Third, we propose the 2D arrangements data structure to model the planar partitions which solves the identified problems related to the maintenance and validity of the planar partitions (*Section* 2.4). Finally, the chapter is concluded with the overall assessment of the proposed data structure (*Section* 2.5).

## 2.2 Representation of Vector Data

A geospatial object described by vector data is composed of three entity: geometry, topology and semantic information. At the lowest level geometry describes the point locations and topology refers to the relationships between the points and higher dimensional geometric primitives such as lines and polygons. Finally, the semantic information is related to any other information that is neither geometric nor topological. Classical data structures like arrays, linked lists and trees can represent the geometric aspects of the geospatial objects but they are not sufficient by themselves for modelling the topological relations explicitly. Specialized and sophisticated data structures might be required depending on the needs. For instance, if data at hand is static, explicit topological information is generally not needed and simpler data structures will do the job. On the contrary, for a dynamic system in which the geometry and topology are constantly changing, explicit topological relations are required for efficient processing and in such cases more elaborate data structures (geometric data structures) have to be employed.

Geometric data structures are not only used in GIS but also in computational geometry and computer graphics. Especially, they are heavily used to represent 2D/3D polygonal mesh models in computer graphics. Different surfaces have different properties and not all data structures are capable of representing all types of surfaces. Three of the most important properties of surface mesh models are described below.

***Manifold surfaces:*** A surface is 2-manifold if each point of the surface is locally homeomorphic (topologically equivalent) to a disk or a half-disk (at the boundaries). This can be interpreted for the case of triangular meshes as: a triangular mesh is 2-manifold if it does not contain a non-manifold edge or a non-manifold vertex nor self-intersecting. A non-manifold edge is incident to more than two faces and a non-manifold vertex is shared by a number of unconnected sets of triangles (*Figure* 2.2).

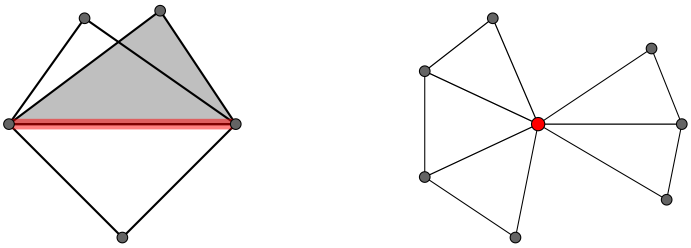


**Figure 2.2:** A non-manifold edge (left) and a non-manifold vertex (right).

***Orientable manifolds:*** A connected 2-manifold mesh is orientable if it is two-sided, meaning that inner and outer sides of the mesh can be distinguishable; otherwise it is non-orientable (one sided). For instance, spherical and cylindrical meshes are orientable whereas Möbius band (*Figure* 2.3) and Klein bottle are non-orientable.

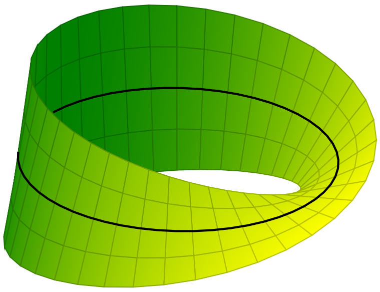


**Figure 2.3:** Möbius band: an example of a non-orientable (single sided) surface.

***Mesh structure:*** The structure of a mesh can be regular, semi-regular or irregular. For instance, in a triangular mesh, a vertex is called regular if it has six neighbouring vertices for the interior vertices and four for the boundary vertices (except for the corners on the boundary). These numbers are four and three respectively for a regular vertex in a quadrangular mesh. A regular mesh is composed of only regular vertices which can be modelled easily (like raster data) with very simple data structures. In addition, the topology of a regular mesh is implicit which can be computed from the regular structure of the vertices. Semi-regular meshes are composed of both regular and irregular vertices and they are generally created by regular subdivision of coarse initial meshes. Irregular meshes, on the other hand, do not show any kind of regularity (*Figure* 2.4).

The knowledge of topological relationships in vector data enables the geometric data structures to traverse fast and efficiently around the local neighbourhood of the geometric primitives [Theobald, 2001]. Topological data structures compute and keep the explicit topological information related to the represented vector data. Therefore, with a topological data structure, topological queries (see *Table* 2.1) can be answered in constant time in average.

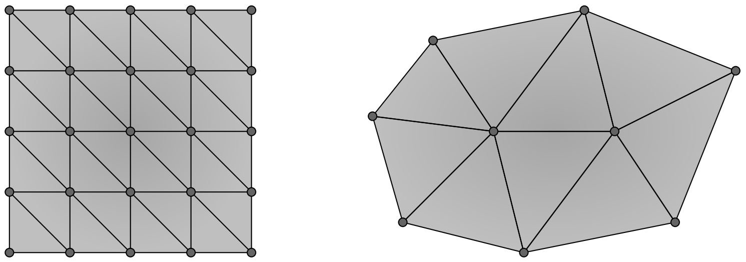


**Figure 2.4:** Surface patches represented with regular (left) and irregular (right) meshes

| **Topological Queries** |
|---|
| - which edges / polygons use this vertex? |
| - which faces (polygons) are adjacent to this face? |
| - which edges border this face? |
| - which vertices belong to this face? |
| - what are the successor (next) and predecessor (before) edges to this edge? |
| - in which face does this vertex lie? |

**Table 2.1:** Examples of topological queries on vector data

In the absence of explicit topological information, topological queries are difficult to answer and at best take linear time depending on the total number of the geometric primitives. Therefore, before selecting a data structure for processing vector data, one has to ask some questions about the structure of the vector data and the algorithmic requirements of the intended application. Is vector manifold or non-manifold, orientable or non-orientable, regular or irregular? Do mesh geometry and topology have to be modified or not? Data structures should be selected based on the answers of these questions.

Different geometric data structures have different approaches to capture the topological relationships. The simplest but more generic geometric data structure called polygonal soup (in computer graphics) or spaghetti (in GIScience) does not store any explicit topological information, therefore it is not suitable for geometric computation. Furthermore, geometric primitives are stored separately without paying attention to shared vertices between the edges or faces. As a result, it is not space efficient but generic (can represent any vector data) and easy to implement. An example definition of polygon soup data structure is given in *Appendix* A.1.

An improvement to the polygon soup data structure is the shared vertex data structure. This data structure does not replicate the vertices, instead it gives references to them from higher-order geometric primitives and reduce the memory requirements. However, topological relationships are still implicit and topological queries take non-constant time with relatively high processing cost. On the other hand, when compared to polygon soup data structure, shared vertex configuration reduces the processing cost when the geometry of the vertices are updated. One possible definition of the shared vertex data structure is given in *Appendix* A.2.

In vector data, vertices are connected to each other by edges; therefore in order to store explicit topological relationships edge-based data structures are needed. Baumgart [1975] developed the first edge-based data structure called winged-edge. In the winged-edge data structure, all edges are directed and the face borders are traversed in clockwise order when looking from outside of the mesh. Moreover, each edge is associated with eight references: two vertices (start and end), two faces (left and right) and four edges (left predecessor and successor, right predecessor and successor) (*Figure* 2.5). Left and right successor and predecessor edges are used for traversing the left and right faces respectively. The name of the data structure comes from these four edges surrounding an edge like a wing. Observe that, edge orientations are not globally consistent. That is, an edge is traversed in opposite directions when its left and right faces are traversed. As a result, the orientation of an edge with respect to the traversed face has to be calculated. Since all connectivity information is kept in edges, for vertices and faces it is enough to store a reference to one of their incident edges. Finally, due to the required orientation information and fixed number of referenced edges, winged-edge data structure can not represent non-manifold and non-orientable meshes. An example definition of the winged-edge data structure is given in *Appendix* A.3.

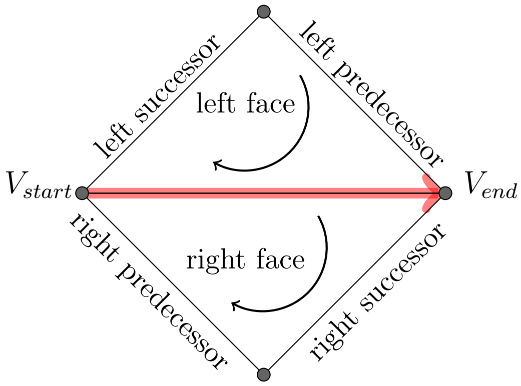


**Figure 2.5:** Winged-edge data structure: An edge is surrounded by four edges (left and right successors and predecessors) forming the winged edge. Faces are traversed in clockwise directions. Edges are oriented but the orientation of an edge changes with respect to the traversed faces.

After the winged-edge data structure, many other edge-based data structures have been developed. The interested reader may refer to Muller and Preparata [1978] for Doubly Connected Edge List (DCEL) data structure, Guibas and Stolfi [1985] for quad-edge data structure, Mäntylä [1988] for half-edge data structure, Weiler [1988] for radial-edges, Campagna et al. [1998] for directed-edges and Kettner [1999] and Botsch et al. [2007] for a review of mesh data structures. *Table* 2.2 summarizes the properties of these data structures.

Among all of these topological geometric data structures, half-edge data structure is quite popular and it has been used a lot both in computer graphics and computational geometry [Botsch et al., 2002]. The half-edge data structure [Weiler, 1985; Mäntylä, 1988] is designed to overcome the orientation problem in the winged-edge data structure. Recall that, calculation of an edge's orientation with respect to the traversed face is necessary for the winged-edge data structure which is inefficient. In the half-edge data structure, each edge is split into two oppositely oriented half-edges called opposite or twin half-edges. Half-edges circulate around the borders of the faces and holes in counter-clockwise and clockwise orientations respectively. For each half-edge, four references are kept: twin

| Data structure | Manifoldness | Orientation | Topological |
|---|---|---|---|
| polygonal soup | both | both | no |
| shared vertex | both | both | no |
| winged-edge | manifold | orientable | yes |
| half-edge | manifold | orientable | yes |
| quad-edge | manifold | both | yes |
| directed-edge | manifold* | orientable | yes |
| radial-edge | both | orientable | yes |

**Table 2.2:** A comparison of some geometric data structures for surface (mesh) representation. *Directed-edge data structure can be extended to support non-manifold meshes.

(opposite) half-edge, next (successor) half-edge, incident face (face on the left) and the end-vertex (target vertex). Moreover, for each vertex a reference is kept for an incident half-edge (one of the out-going half-edges) and similarly for each face a reference for an incident half-edge (one of the half-edges circulating around the face in counter-clockwise direction) is stored (*Figure* 2.6).

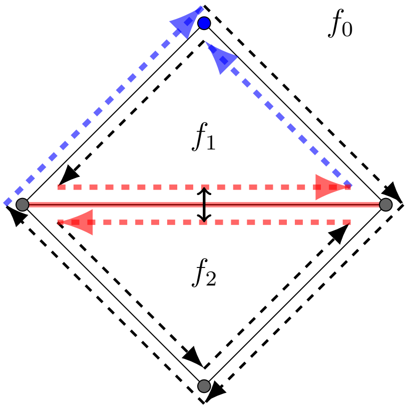


**Figure 2.6:** Half-edge data structure: faces and holes are circulated by half-edges in counter-clockwise and clockwise order respectively. Observe that the union of $f_1$ and $f_2$ is a hole in the unbounded face $f_0$. Moreover, blue half-edges are incident to the blue vertex and two red half-edges are twin representing the same edge. Faces store a reference to one of the circulating half-edges and an half-edge stores references to target vertex, the face on its left, twin half-edge and next half-edge.

The half-edge data structure is also known with different names in the literature. Weiler [1985] used FE-structure, Mäntylä [1988] used the half-edge as in this dissertation and de Berg et al. [1997] used DCEL although it is a different data structure in the original paper defining the DCEL [Muller and Preparata, 1978]. An example definition of the half-edge data structure is given in *Appendix* A.4.

In computer graphics, Stereolithography (STL) file format ([STL, 1989]) utilizes the polygon soup data structure for storing mesh models. Moreover, Object File Format (OFF), Wavefront obj and Virtual Reality Modelling Language (VRML) file formats use the shared vertex data structure. On the other hand, each topological geometric data structure should have its own format for storing and loading mesh models. In GIScience, many

GIS formats follow the Open GIS Consortium (OGC) simple features standard [OGC Simple Features, 2011] and utilize the polygonal soup or spaghetti data structure. The most famous of these formats is the Environmental Systems Research Institute (ESRI) shapefile [ESRI Shapefile, 1998]. Topological data structures have also been used in GIS, most notable examples are GRASS GIS[1] and PostGIS Topology[2].

GRASS GIS has a native format called vector maps for representing 2D and 3D[3] topological vector data. A vector map is composed of low level primitives: point, line, boundary, centroid, face and kernel. These primitives are used to construct higher level topological structures called area, isle, volume and hole. In addition, there exist some internal rules that apply to vector data such as boundaries should not cross each other (lines can cross each other), lines and boundaries can only share endpoints, areas must be explicitly closed, common area boundaries should appear only once, etc. In GRASS GIS, the aforementioned primitives and the higher level constructions on them are encapsulated within a series of class hierarchy together with the rules to manage them. In addition, there exist modules for processing the vector data for computing the topology of a non-topological vector data, topological editing, automatically fixing topological errors (to some extend), etc.

PostGIS Topology package was started with version 2.0 (released in April 2012) and it provides a topological database allowing its users to construct a topology schema out of three fundamental topological primitives: faces, edges and nodes. It also includes accessory functions[4] for creating, editing and validating the topological vector data.

## 2.3 Planar Partitions and Planar Graphs

Planar partitions and planar graphs are strongly related to each other. In fact, every planar partition defines a planar graph. Graph theory has been studied since 18th century and it is a well established domain in mathematics and computer science. In GIScience, researchers are well-aware of this fact and use planar graphs to analyse planar partitions [Plümer and Gröger, 1997]. Before discussing planar partitions in detail, it is useful to remember the basic definitions for graphs and planar graphs.

### 2.3.1 Planar Graphs

Graphs are mathematical notions used to model pairwise relations between objects from a certain set. A graph $G = (V, E)$ consists of two sets, $V$ and $E$ such that the elements of $V$ and $E$ are called vertices (singular: vertex) or nodes and edges, respectively. Each edge represents a connection between a pair of vertices (*Figure* 2.7).

There are many different types of graphs. For example, if the vertex pairs forming the edges are ordered, the graph is a directed graph (digraph); if the cardinality of the vertex

[1] https://grass.osgeo.org/

[2] http://postgis.net/docs/manual-dev/Topology.html

[3] The topological support for 3D vector data is partial and not fully functional at the time of writing for GRASS GIS version 7.1.

[4] PostGIS Topology package partially relies on the GEOS library (https://trac.osgeo.org/geos/) to provide the related data structures and various topological accessory functions.

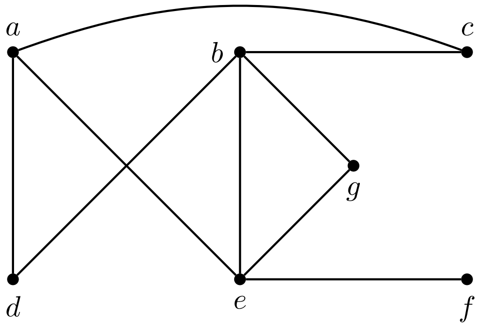


**Figure 2.7:** An example graph: $G = (V, E)$, $V = \{a, b, c, d, e, f, g\}$ and $E = \{\{a, c\}, \{a, d\}, \{a, e\}, \{b, d\}, \{b, e\}, \{b, g\}, \{e, g\}, \{e, f\}\}$.

set is infinite, then the graph is infinite; if multiple edges are allowed between two vertices, then the graph is multi-graph, etc. [Gross et al., 2014; Harris et al., 2008]. In addition, a graph embedding is a particular drawing of a graph on a surface (planar, spherical, hyperbolic, etc.). A graph may have exponential number of embeddings on a given surface [Battista et al., 1998]. The most commonly encountered graph embeddings are generally straight line drawings, in which all graph edges are drawn as straight line segments. A Graph $G = (V, E)$ is said to be planar if it can be drawn on a plane in such a way that pairs of edges intersect only at the vertices of the graph, if at all. If $G$ has no such embedding, then G is non-planar. The graph given in *Figure* 2.7 is a planar graph but it is not drawn with planar embedding. The same graph can also be drawn as in *Figure* 2.8 which is a planar embedding.

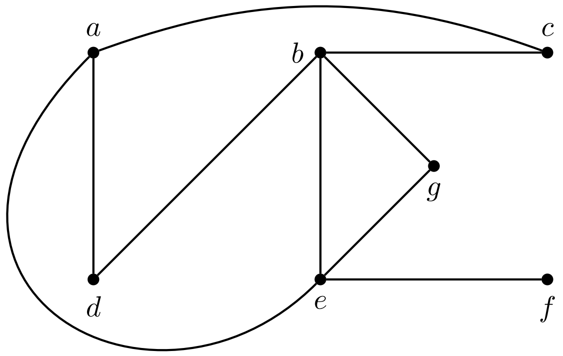


**Figure 2.8:** Redrawn of the planar graph in *Figure* 2.7 with a planar embedding

A planar embedding of a planar graph partitions the supporting plane into regions called faces and the unbounded region is called the outer face. Leonhard Euler discovered a beautiful formula which describes the relations between the number of vertices ($|V|$), edges ($|E|$) and faces ($|F|$) of connected planar graphs. Euler's formula is:

$$|V| - |E| + |F| = 2 \tag{1}$$

Observe that, for the planar graph in *Figure* 2.8, $|V| = 7$, $E = 9$, $F = 4$ and Euler's formula holds ($7 - 9 + 4 = 2$). Another useful formula derived from Euler's formula is:

$$|E| \leq 3|V| - 6, \quad |V| \geq 3 \tag{2}$$

The $K_5$ graph given in *Figure* 2.9 is an example of a non-planar graph. Notice that, no matter how hard one can try, it cannot be untangled and embedded into a plane. Notice also that, $K_5$ has 5 vertices and 10 edges which conflicts with the *equation* (2). However, *equation* (2) cannot be used for all graphs to test planarity; there exist some graphs which are not planar but satisfy the *equation* (2). Such a graph is given in *Figure* 2.10. Thus, proving planarity may be tricky if a quick planar embedding cannot be found and the size of the graph is relatively high. There are a number of methods to prove whether a given graph is planar or not, please refer to Boyer [2004] for one of them.

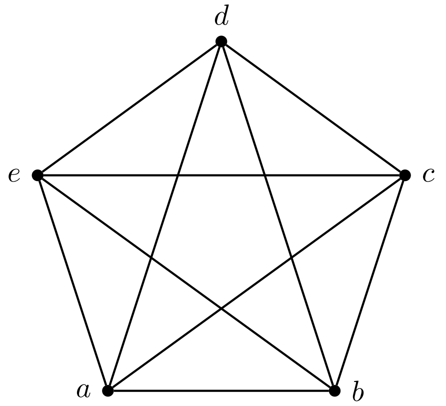


**Figure 2.9:** An example of a non-planar graph ($K_5$)

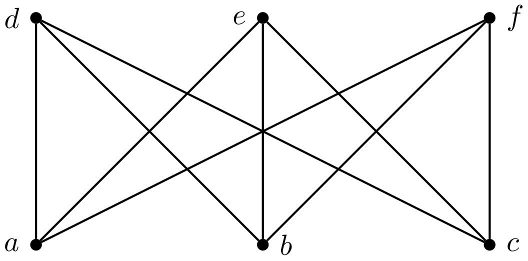


**Figure 2.10:** An example of a non-planar graph ($K_{3,3}$)

## 2.3.2 Planar Partitions

A planar partition is the tessellation of a planar surface with a set of non-overlapping polygons (*Figure* 2.11). Planar partitions are frequently used in GIS to represent thematic and geometric spatial information such as land use, vegetation cover, cadastral parcels and administrative boundaries [Penninga et al., 2005; Ohori, 2010; Ledoux and Meijers, 2010]. An example dataset can be Coordination of Information on the Environment (CORINE) land cover provided by European Environmental Agency (EEA) for mapping land use in Europe.

In GIScience, planar partitions are often represented and stored as a set of individual polygons following the simple features specification [OGC Simple Features, 2011]. Although such a representation is very simple and requires no complex data structures, it is vulnerable to errors that can be introduced when the planar partitions are built or modified. In fact, topological and geometric errors are common in planar partitions such

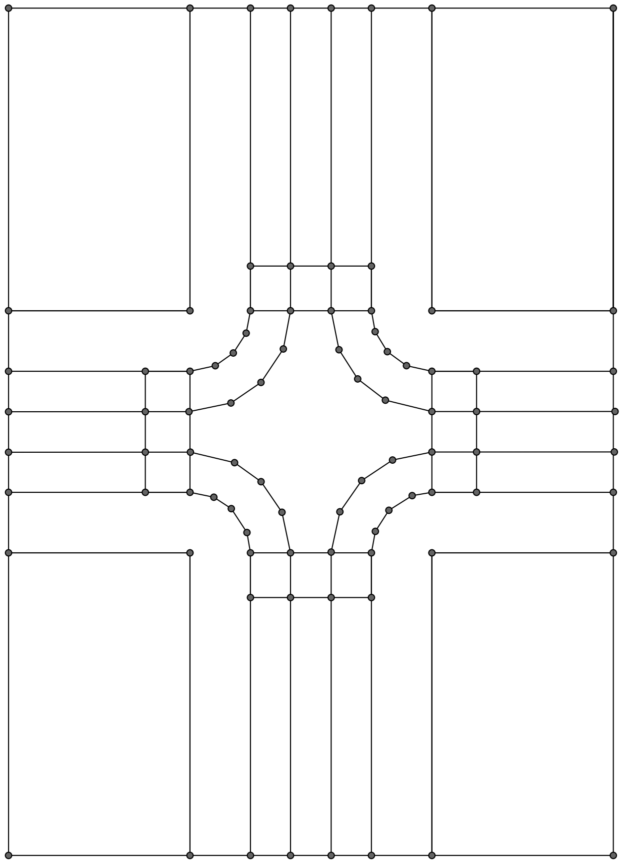


**Figure 2.11:** An example planar partition that models a street scene. The surface of the street is partitioned based on the footprints of the urban objects: buildings, sidewalks, pedestrian crossings, road lanes and an intersection. Notice that, this planar partition is also a planar graph embedded in a plane. That is, all the graph vertices lie on the same plane and the edges of the graph do not intersect other than vertices.

as overlapping polygons, gaps between polygons and erroneous polygons. The source of these errors can be: usage of inexact arithmetic, limited machine precision, and human operator errors if data is modified manually [Ohori, 2010; Ledoux and Meijers, 2010].

A workflow of managing planar partitions is generally composed of three steps: creation, validation and maintenance. For some applications only the first two steps can be sufficient if the data is static. On the other hand, for dynamic systems in which the data is continuously changing, the underlying planar partitions have to be maintained valid (*Figure* 2.12). In this, section we will have a look at these work flow steps, and discuss how they can be achieved using GISs and in the following section (*Section* 2.4), the same pipeline will be analysed when 2D arrangements are used to represent planar partitions.

Planar partitions can either be created from scratch by using a GIS or previously created datasets can be utilized. However, the datasets obtained by combining multiple existing datasets rarely constitute a planar partition due to the geometrical and topological errors mentioned in the previous paragraph and mismatches also known as conflation problems [Lynch and Saalfeld, 1985; Yuan and Tao, 1999]. Refer to *Section* 6.4.1 for a planar partition creation experience from OpenData Paris[5] dataset.

By definition, a valid planar partition has to comply with the following constraints given

[5] http://opendata.paris.fr/page/home/

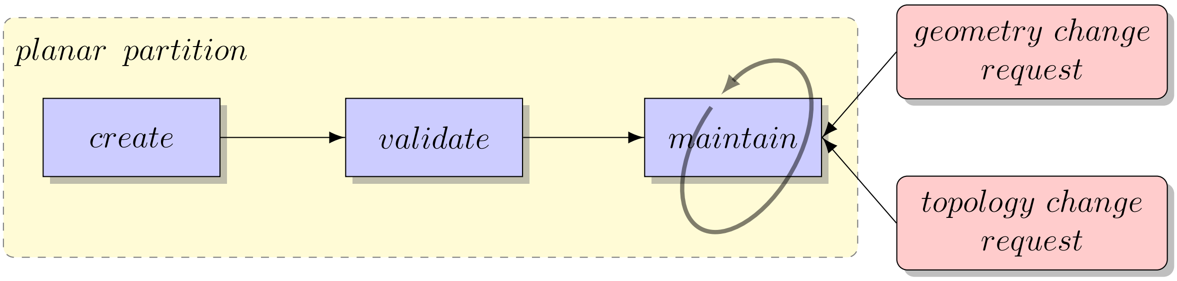


**Figure 2.12:** A typical workflow for applications managing planar partitions

in the *Table* 2.3.

| No | Constraints |
|---|---|
| 1. | All polygons should be valid. |
| 2. | Polygon interiors should not intersect. |
| 3. | There should be no gaps between the polygons. |

**Table 2.3:** Constraints for a valid planar partition

In OGC Simple Features [2011], a polygon is defined as follows: a polygon is a planar surface defined by one (1) exterior boundary (exterior ring) and zero (0) or more inner boundaries (interior rings). Each interior boundary defines a hole in the polygon. Exterior rings traverse the boundary in a counter-clockwise direction and interior rings have the opposite orientation. Furthermore, a set of constraints on the validity of polygons were defined as shown in *Table* 2.4 and *Figure* 2.13 displays some examples of invalid polygons violating the constraints.

| No | Constraints |
|---|---|
| 1. | Polygons are topologically closed. |
| 2. | The boundary of a polygon consists of a set of linear rings that make up its exterior and interior boundaries. |
| 3. | No two rings in the boundary cross and the rings in the boundary of a polygon may intersect at a point but only as tangent. |
| 4. | A polygon may not have cut lines, spikes or punctures. |
| 5. | The interior of every polygon is a connected point set. |
| 6. | The exterior of a polygon with one (1) or more holes is not connected. Each hole defines a connected component of the exterior. |

**Table 2.4:** Constraints for a valid polygon according to OGC Simple Features [2011].

GISs such as ArcGIS[6], Java Topology Suite (JTS)[7], GRASS GIS, QGIS[8] (topology checker plugin) and PostGIS Topology have tools for validating and fixing individual polygons by checking a list of constraints similar to given in *Table* 2.4 if not the same. Assuming that all the polygons in a planar partition are valid, there may exist two types of invalid configurations within the planar partition: gaps between the polygons and polygon overlaps.

[6] http://www.esri.com/software/arcgis

[7] http://tsusiatsoftware.net/jts/main.html

[8] http://www.qgis.org/en/site/

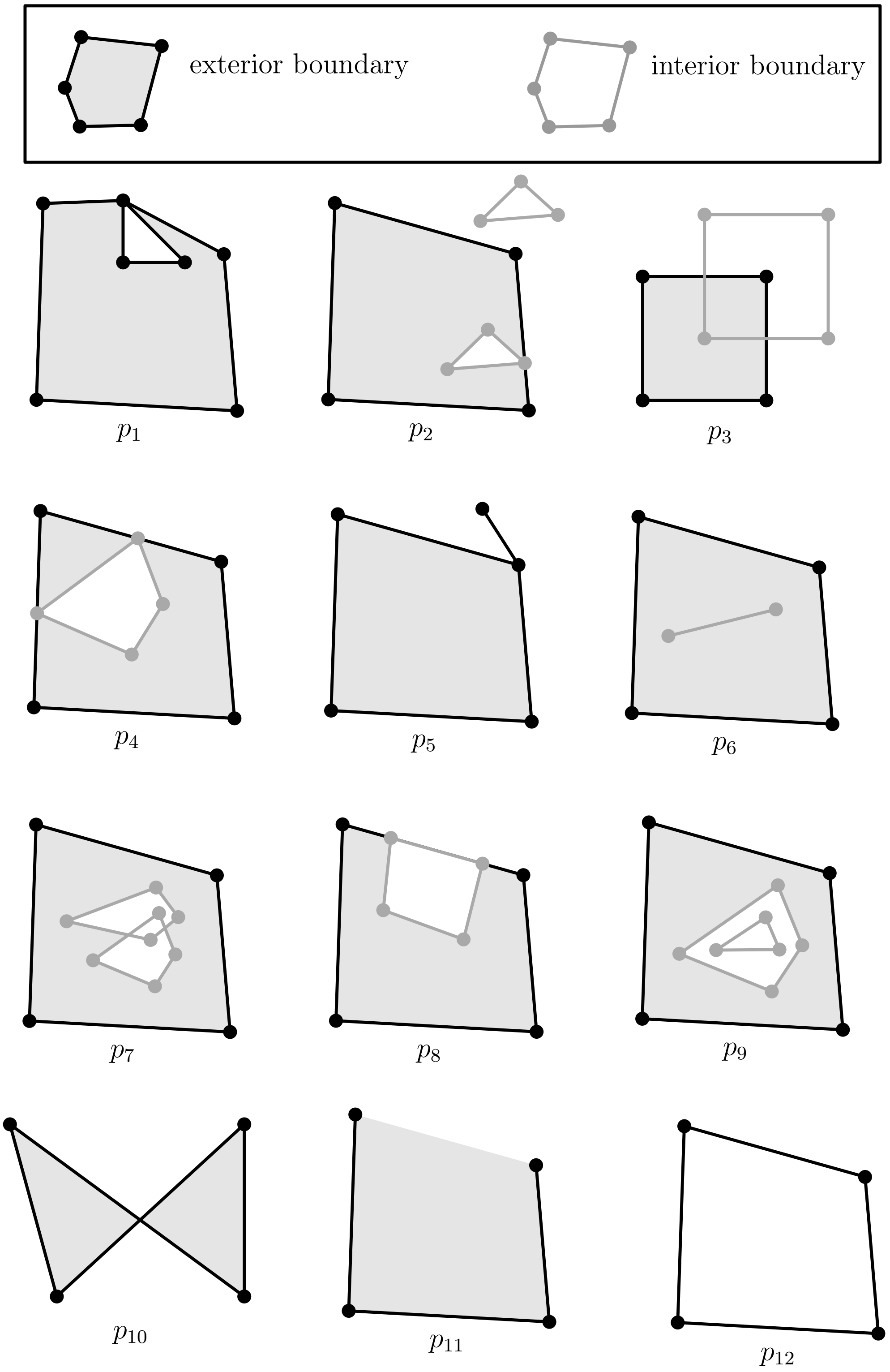


**Figure 2.13:** Examples of invalid polygons (taken from [Ledoux et al., 2012]) according to OGC Simple Features [2011] polygon definition and its constraints given in *Table* 2.4: $p_1$ violates the polygon definition by having more than one exterior ring, $p_2$ has an interior ring outside the exterior ring, $p_3$, $p_7$, and $p_8$ have crossing rings (constraint 3), $p_4$ partitions the interior of the polygon into non-connected sets (constraint-5), $p_5$ and $p_6$ violates the constraint 4, $p_9$ has two interior rings one inside another, $p_{10}$, $p_{11}$ and $p_{12}$ violate constraints 3, 1 and 5 respectively.

Unlike checking the validity of individual polygons, these last two conditions require the knowledge of topology around the polygons. Without topological information, checking these conditions would be computationally very expensive (checking any pair of polygons are disjoint or not and finding the gaps within the union of the polygons that are distinct from the holes). In fact, for fixing and validating planar partitions, ArcGIS allows users to define a set of topological rules applying to the processed datasets such that overlaps and gaps between the polygons can be detected [ESRI, 2003]. Similarly, GRASS GIS and PostGIS Topology also possess topological information which can be used to check the related validity conditions. For instance, PostGIS Topology has a method called "ValidateTopology" in which the overlapping polygons are detected. There isn't any direct function in PostGIS Topology for detecting gaps but in a topological database this can be achieved by using a sequence of other topological operations. In addition, GRASS GIS has also a similar function called "v.clean" performing topological fixes. In fact, many GISs have snapping functions which automatically connects vertices to vertices, vertices to edges and edges to edges based on threshold values. These tools can also be used to validate or fix errors in the planar partitions.

Apart from commercial or well-known open source GISs, Ohori et al. [2012] presented two tools for fixing polygons (prepair) and planar partitions (pprepair) depending on constrained triangulations. Moreover, Plümer and Gröger [1997] considered planar partitions as planar graphs and they defined seven axioms (*Table* 2.5) for validating planar partitions. That is, planar graphs satisfying the provided axioms are valid planar partitions (interested readers may refer to the corresponding article for the proofs). The axioms are mathematical and hence relatively easier to check with computers which provides another tool for validating the planar partitions apart from employing the planar partition definition. Matijevic et al. [2008] utilized these axioms to maintain the validity of planar partitions in an online database environment.

| No | Axioms |
|---|---|
| 1. | For each vertex in the topology relation there is exactly one vertex in the geometry relation (referential integrity). No two different vertices have the same coordinates (uniqueness). |
| 2. | Each vertex has at least two incident edges (vertex degree $\geq 2$). |
| 3. | For every edge there are exactly two distinct vertices as end vertices. |
| 4. | Edges correspond geometrically to straight line segments. No two such segments share any point except at their ends (non-intersecting edges). |
| 5. | Each edge has exactly two incident faces. |
| 6. | Each face has exactly one simple cycle as window. |
| 7. | No midpoint of an edge lies in the interior of a face. |

**Table 2.5:** Axioms for testing the validity of planar partitions [Plümer and Gröger, 1997]. Axioms are related to the vertices (1, 2), the edges (2, 3, 4) and the faces (6, 7) of the planar graphs. Axiom 1 and 4 imply the planarity of the graph, axiom 2 implies connectivity of the graph and prevents dangling edges, axiom 3 prevents zero-length edges, axiom 5 avoids gaps, axiom 6 requires simple polygons (non self-intersecting) corresponding to the faces and axiom 7 prevents overlaps between the polygons.

The validation process within the workflow given in *Figure* 2.12 has to be re-run after each modification to planar partitions. Therefore, in a dynamic system managing planar partitions, the efficiency of the validation or the maintenance step mostly defines the overall performance of a GIS.

## 2.4 2D Arrangements as Planar Partitions

Let a set of curves $C = \{c_1, c_2, ..., c_n\}$ embedded on a surface $S$ is given. The elements of $C$ divide $S$ into a finite number of cells of dimension 0 (vertices), 1 (edges) and (2) faces. This subdivision is the arrangement (denoted by $A(C)$) induced by $C$ on $S$ [Agarwal and Sharir, 1998]. Therefore, depending on the type of $S$ (e.g., plane, cylinder, sphere, parametric surface, etc.) and the types of curves in $C$ (e.g., lines, conic sections, polynomial curves, rational curves, etc.) the characteristics of the arrangement changes. Furthermore, the curves can be either bounded or unbounded. In this dissertation, we are interested in the arrangements of 2D line segments[9] on the plane for representing planar partitions. We also utilized arrangements of linear and quadratic Bézier curves for modelling the centrelines of walkways which are discussed in *Chapter* 6. Interested reader may refer to Berberich et al. [2010b,a] for arrangements on parametric surfaces and Fogel et al. [2008] for arrangments of geodesic arcs. In addition, Agarwal and Sharir [1998] provided a broad survey of applications, algorithms and representations for arrangements. Finally, apply to Edelsbrunner et al. [1986] for an algorithm to generate arrangements of lines on a plane.

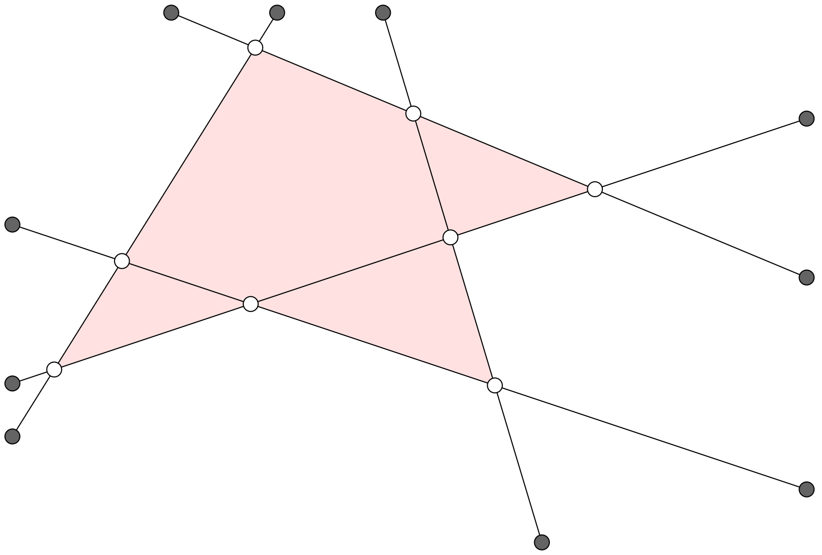


**Figure 2.14:** An example of a 2D arrangement induced by 5 line segments: the arrangement is composed of 5 faces one of which is the unbounded face, 18 vertices (8 of them are generated from segment intersections and 10 of them are segment end-points), and 21 edges. Observe that the arrangement is also a planar graph satisfying equation 1 (Euler's formula) (18 - 21 + 5 = 2).

Observe that, a 2D arrangement on the plane is a straight line embedding of a planar graph according to the planar graph definition given in *Section* 2.3.1, i.e. each vertex of the arrangement is on the plane and edges do not intersect other than vertices (*Figure* 2.14). As a result, 2D arrangements can be tested with respect to the planar partition axioms defined in *Table* 2.5. The definition of the 2D arrangements automatically satisfies

[9] In this dissertation, a 2D arrangement refers to an arrangement of 2D line segments unless otherwise is stated.

six of the seven axioms other than the axiom-2. Therefore, in order to determine whether a given 2D arrangement is a valid planar partition or not, testing the arangement with the axiom-2 is sufficient. That is, a 2D arrangement without isolated vertices and dangling edges[10] is a valid planar partition. Similarly, the constraints for valid polygons given in *Table* 2.4 are also automatically satisfied by definition except the constraint-4 which is directly related to the axiom-2 in *Table* 2.5. In addition, no gap or overlap is allowed within a 2D arrangement by definition. Consequently, the constraints given in *Table* 2.3 are also reduced to the same single constraint (constraint-4 in *Table* 2.4 or axiom-2 in *Table* 2.5).

The workflow of a GIS that manages dynamic planar partitions has already been shown in *Figure* 2.12. Representing planar partitions with 2D arrangements does not affect the first step of the workflow. Planar partitions still have to be created either from scratch or leveraging the existing vector datasets using a GIS. However, the second (validation) and third (maintenance) steps in the work flow are more easier to manage with 2D arrangements representation. Only the 2D arrangement data structure has to be maintained upon adding or deleting geometry. That is, arrangement cells (vertices, edges and faces) have to be created, deleted and split as the arrangement geometry is being modified and the topological relationships among the cells have to be updated accordingly. When the planar partitions or individual polygons are needed, they can easily be extracted from 2D arrangements by applying the axiom-2 of *Table* 2.5 (*Figure* 2.15).

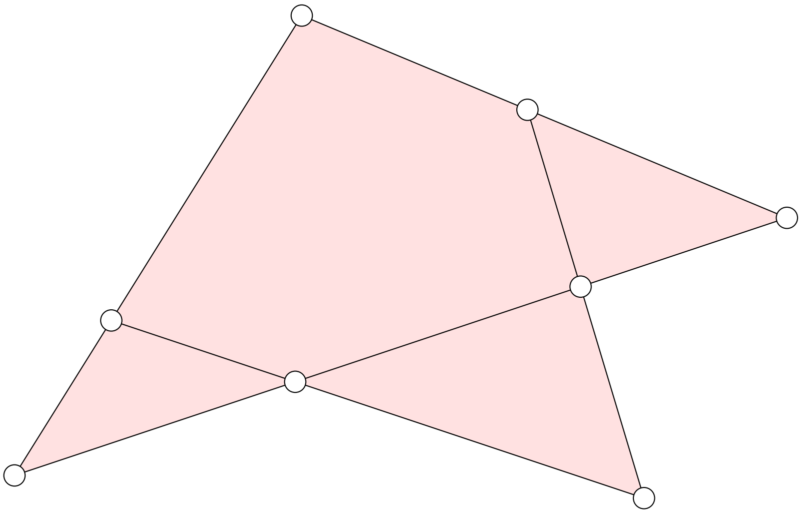


**Figure 2.15:** Extracting a planar partition from an arrangement of line segments: for this example (also see *Figure* 2.14), eliminating antenna like structures by removing the vertices with *degree* < 2 and the connected edges to them, is enough to get a planar partition.

Arrangements require the calculation of curve intersections that are embedded on a surface. These intersection points together with the curve endpoints constitute the vertices of an arrangement. Furthermore, isolated vertices can also be allowed to be inserted in an arrangement for representing punctual features on a surface. The edges of an arrangement are composed of the curve segments in-between two vertices and the faces are regions

[10] The number of incident edges to a vertex defines its degree. Isolated vertices are of degree 0 and dangling edges are incident to vertices of degree 1.

bounded by edges. Arrangements described by these cells (vertices, edges, faces) need a data structure to manage the cells. Although, it can be any data structure, a topological data structure allows an efficient and effective geometric computing (e.g. traversal of the cells, queries on the arrangement such as point locations) as discussed in *Section* 2.2.

### 2.4.1 Implementation Details

We utilized the 2D Arrangement package [Wein et al., 2015; Fogel et al., 2012] of The Computational Geometry Algorithms Library (CGAL)[11] for implementing the 2D arrangement represented planar partitions. In this package, 2D arrangements are represented with the half-edge data structure (*Figure* 2.16). We further attached a Universally Unique Identifier (UUID) [ITU-T X.667, 2012] to each vertex, edge and face at the time of creation. These identifiers are used to create permanent links (serializable identifiers which are stable upon editing, load and save) between the semantic objects (see *Chapter* 3) and the cells of the arrangements.

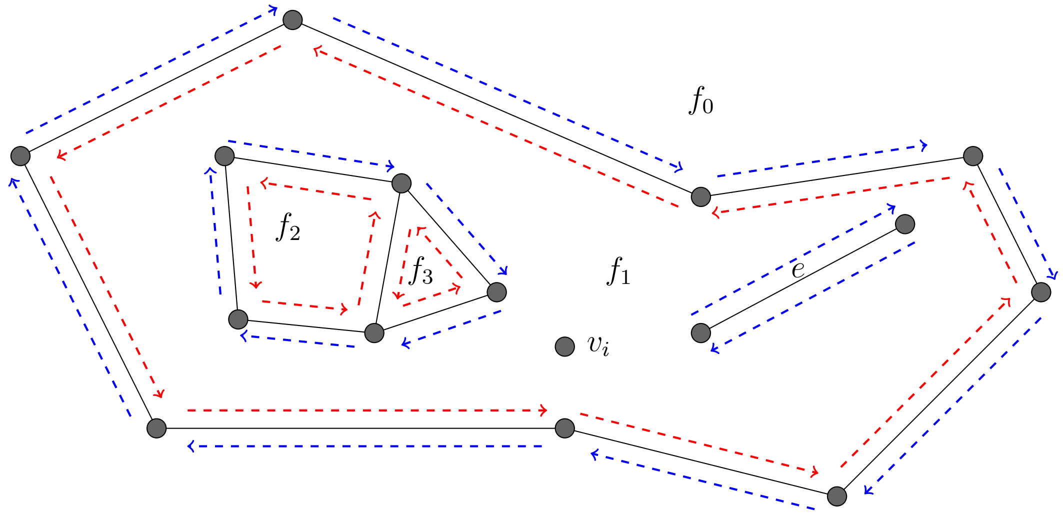


**Figure 2.16:** A 2D arrangement and its representation with the half-edge data structure: each edge is represented by oppositely oriented half-edges (twin half-edges). Half-edges circulate around faces and holes in counter-clockwise (red) and clockwise (blue) directions respectively. Each half-edge is incident to the face on its left and to the vertex to which it is directed. $f_0$ is the unbounded face containing $f1$ as a hole. $f_1$ has two holes (union of $f_2$ and $f_3$, edge $e$) and an isolated vertex ($v_i$). Faces keep track of holes via one of the circulating half-edges around each hole. In addition each face stores a list of isolated vertices that lie in the face and equally each isolated vertex has a link to its containing face.

Moreover, CGAL's 2D Arrangement package also has a nice feature which allows point location queries within the arrangement. The return from the query is a reference to a vertex, an edge or a face on which the query point lies. In addition, the package also has a vertical ray shooting feature in which a ray is shot vertically (up or down) from a point and the first arrangement cell hit by the shoot is returned. These features are used in the user interface for snapping/selecting to arrangement cells (by querying the mouse pointer within the arrangement) and semantic objects constructed upon them.

[11] http://www.cgal.org/index.html

## 2.5 Conclusion

In this chapter, we have discussed different representations of vector data with different data structures for storing and processing them in computers. We mentioned that although some simple data structures are good enough for specific purposes, topological data structures are required for efficient and effective geometry and topology processing. Being a type of vector data and used in many GISs as well as in ours, the rest of the chapter is devoted to the analysis of planar partitions and their representations in GISs.

We proposed to represent planar partitions with topological arrangements of 2D line segments which are in fact modelling planar graphs (not necessarily the connected ones). We have seen that the definition of 2D arrangements satisfies all of the mathematical axioms except one which were developed by Plümer and Gröger [1997] (*Table* 2.5) for checking the validity of the planar partitions. The remaining axiom is related to the isolated vertices and dangling edges which might also be useful for some applications (e.g. we used isolated vertices to create punctual objects corresponding to the building doors in our pedestrian network generation application). In addition, this last axiom can be applied to 2D arrangements in order to filter out these additional primitives to obtain pure valid planar partitions whenever they are needed.

It has already been stated in the chapter that creation, validation and maintenance related tasks of planar partitions can also be performed using different desktop or database GISs. Each GIS has its own data model and implementation that suit to the reasons of its existence. It would be a difficult task to compare of all these GISs according to their performance (possibly with different perspectives) in managing planar partitions. After stating this, the author would like to comment conceptually on the performance of the proposed planar partition representation. 2D arrangements (borrowed from computational geometry) implemented with a dedicated topological data structure provides a compact and efficient way of encoding the aforementioned mathematical axioms for checking the validity of planar partitions. Besides, the axioms are encapsulated within the core data structure which hides many issues from the application level services. In addition, applying the exact computation paradigm for robust and exact geometric computation (see *Section* 5.5) is relatively easier with such a compact system which is not possible to obtain in many GISs.

2D line segment arrangements are limited to 2D planar surfaces. Therefore, it is not possible to model outlines of geospatial objects that are overlaid on top of each other without sharing the same physical surface (e.g. 3D roads passing on top of each other and bridges over rivers). In order to support such cases, 2D arrangements can be lifted to 2.5D by storing an additional height value into the arrangement cells. This height value should not be considered as a real third coordinate that lifts up the cells into 3D. It can rather be thought as a variable which indicates the existence of multiple arrangement cells whose vertical projection coincide (*Figure* 2.17).

For real 3D arrangements, 3D (topological) data structures are required. An example to a such data structure can be the combinatorial / generalized maps [Lienhardt, 1991; Françon and Bertrand, 2000; Damiand and Lienhardt, 2015] which can represent n-dimensional subdivided objects with a topological boundary representation. However, algorithmically and computationally difficult part of 3D arrangements is the calculation of boolean set operations (intersection, union and difference) on the 3D geometric primitives (arbitrary

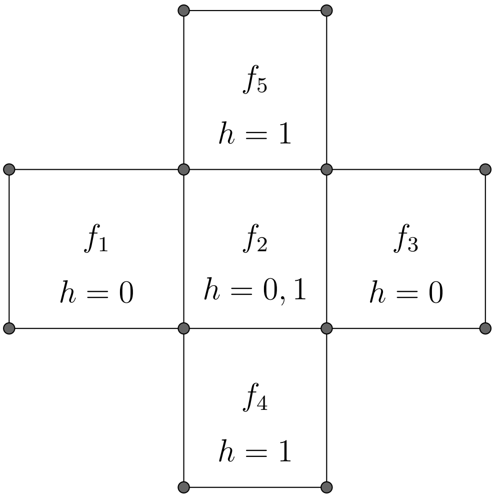


**Figure 2.17:** 2.5D arrangements can be used to solve the bridge problem. Observe that the face in the middle ($f_2$) has two associated height values indicating that there exist two faces on top of each other at different height values. In this example, the height information is stored to the arrangement faces. The height information stored to any arrangement cell can be inherited by the incident lower dimensional cells.

3-polytopes, 3D line segments, 3D surface patches, etc.).

# Chapter 3

# Object-based GIS Modelling

## 3.1 Introduction

The data used in GIScience (geo-data) contain both thematic and geometric (spatial) information which need to be merged for many GIS applications. Mainly, there exist two approaches for linking thematic and geometric aspects of geo-data: field approach and object oriented approach [Molenaar, 1998] (*Figure* 3.1). In the first approach, thematic data are linked to geometry (points or finite cells) using attributes. In this structure, attribute values are position dependent and they have to be calculated for each point or cell. The second approach organizes data in an object oriented model by defining objects, each of which has a location, a shape and several non-geometric characteristics.

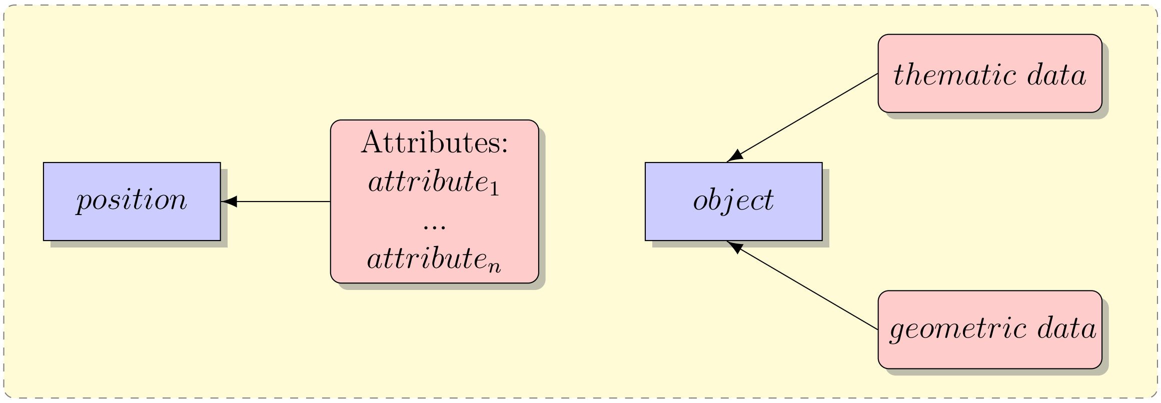


**Figure 3.1:** Two approaches for linking thematic and geometric information: field model (on the left) stores thematic information as attributes to geometric positions or cells whereas object oriented model (on the right) structures the data into objects. This figure is re-composed from Molenaar [1998].

Human-beings tend to think thematically, therefore thematic aspects of data are often of prime importance. This means that the data querying and processing will be organised and formulated primarily from a thematic perspective. Hence, the structuring and formation of the geometric data depend on the thematic aspects of the data [Huxhold, 1991]. According to this, object oriented approach might be more desirable than the field approach since object's are classified and grouped thematically. In addition, object oriented approach utilizes all the concepts of object oriented modelling and programming such as encapsulation, inheritance, data abstraction, etc. Object oriented approach has been used in numerous works such as Egenhofer and Frank [1992]; Günter and Lamberts [1994]; Gong and Li [1996]; Gonorov and Khorev [1996]; Tryfona et al. [1997], etc.

In *Chapter* 2 planar partitions has been discussed for modelling the space occupancy of urban objects in public scenes. Recall that we utilized 2D arrangements for managing the geometric aspects of the spatial data. In this chapter our focus is on the thematic / semantic aspects of the data with an object oriented approach.

## 3.2 Semantic Objects and Semantic Planar Partitions

In computer science, an object is a buffer of memory whose address is referenced by an identifier (a name, reference or pointer). Software objects are used to model the state and behaviour of real-world objects in computing environments. In GIScience, software objects are generally used to model geospatial objects such as buildings, streets, roads, walkways, parks, city furnitures, lampposts, etc. In our system, we call such objects as semantic objects[1] referring to both the geometry and the semantic information stored in an object.

Semantic objects are constructed on top of the geometry stored in 2D arrangements such that every semantic object is linked to a set of arrangement cells (*Figure* 3.2). Generally, the cells with highest dimension are used for this mapping. For instance, to link a polygonal region to an object, only the corresponding face of the arrangement is used. The edges and vertices composing the boundary of the face are not directly linked to the object[2]. In reality, there is no restriction on these mappings. Moreover, semantic objects can also be created on top of each other which forms an object hierarchy. In an object hierarchy, parent objects may also be directly linked to the underlying geometry. Indeed, semantic objects can be classified according to how they are matched to the underlying 2D arrangement cells and other objects (*Table* 3.1).

| Object Type | Description |
|---|---|
| areal | objects constructed only from the arrangement faces. |
| linear | objects constructed only from the arrangement edges. |
| punctual | objects constructed only from the arrangement vertices. |
| mixed | objects constructed from different types of arrangement cells. |
| level-0 | objects that are directly and only liked to arrangement cells. |
| parent | objects composed of other objects. |
| composite | objects composed of other objects and arrangement cells. |
| disjoint | when the object geometry is disjoint. |
| connected | when the object geometry is connected. |

**Table 3.1:** Semantic object types according to geometry properties of the objects, i.e. how they are connected to the cells of the 2D arrangement and other objects.

[1] From now on, the word "object" should refer to "semantic object" although it is not explicitly mentioned.

[2] Incidence relationships can be used to access the lower dimensional geometric primitives of an object through the higher dimensional primitives.

In our system, as soon as semantic objects are created on top of the planar partitions, we call them semantic planar partitions. *Figure* 3.2 displays an example of a semantic planar partition.

## 3.3 Object Hierarchy

The hierarchy between the semantic objects is constructed using a Directed Acyclic Graph (DAG). DAGs are directed graphs that are free of cycles. DAGs arise naturally when the graph nodes often have a natural ordering e.g. when the nodes represent events ordered in time or objects ordered by hierarchy [Gross et al., 2014].

The leaf nodes (sink nodes[3]) of the hierarchy graph do not represent the semantic objects but the geometric primitives (cells of the corresponding arrangement) that are used to construct the semantic objects. The lowest level objects (level-0 objects) are in fact represented in the hierarchy with the level-1 graph nodes. The source nodes[4] of the hierarchy graph correspond to the top level objects in the object hierarchy which are in general, objects referring to logical constructions in cities such as avenues, streets, regions or even the cities themselves.

*Figure* 3.3 displays the hierarchy of the objects constructed for the semantic planar partition represented in *Figure* 3.2. A sub-graph of the hierarchy graph is also given in *Figure* 3.4 for a higher resolution zoomed view.

## 3.4 Maintenance of Semantic Planar Partitions

In *Section* 2.3.2, we have discussed the maintenance of the planar partitions for keeping them valid upon geometric or topological updates. Similarly, semantic objects also need to be maintained. Adding geometry to the underlying arrangements and deleting geometry from them require different actions to be taken.

When level-0 semantic objects are created, links from the objects to the related arrangement cells are constructed. After an insertion of new geometry into the arrangements, some of the arrangement cells will be modified. Some faces and edges might be split and as a result new cells are created. If an arrangement cell that has been linked to a semantic object is updated (i.e., split), the corresponding semantic object has to be notified. Upon notification, the links to the arrangement cells from the notified object has to updated such that the object's initial geometry does not change. *Figure* 3.5 displays an example for the maintenance of the semantic objects when a line segment is added to the underlying arrangement. Note that in the object hierarchy, only the level-0 and composite objects need to be maintained after new geometry addition/insertion.

On the contrary to the addition of geometry, deletion of geometry is constrained to a set of rules. The non-linked arrangement cells[5] and the cells that are not incident to a linked cell can be freely deleted from the arrangements. On the other hand, the remaining arrangement cells (i.e. the linked cells and the cells that are incident to them) cannot

[3] Sink node: a graph node with out-degree equal to 0.

[4] Source node: a graph node with in-degree equal to 0.

[5] Non-linked cell: An arrangement cell that is not linked to a semantic object.

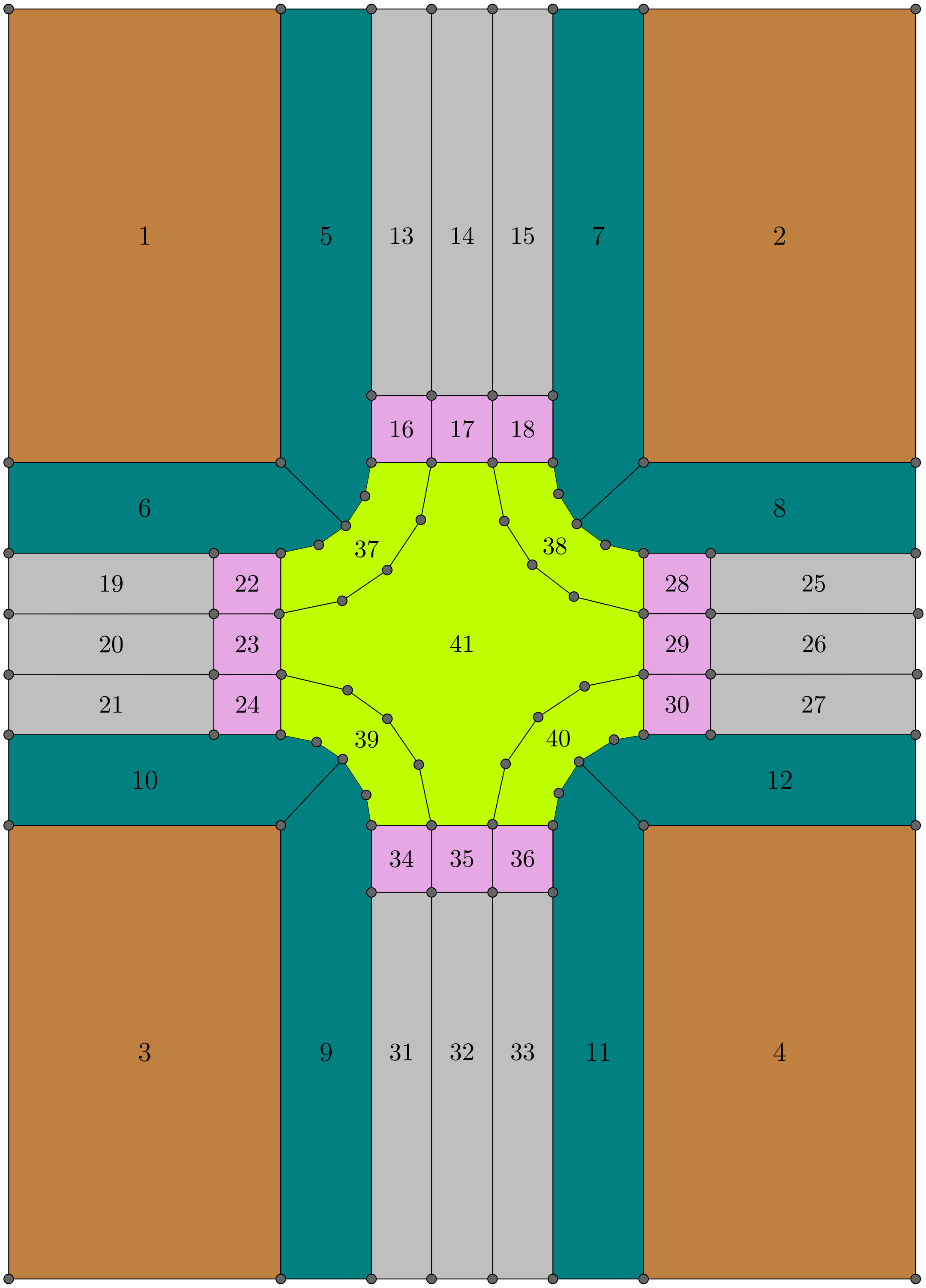


**Figure 3.2:** An example of a semantic planar partition: semantic objects are created on top of a planar partition modelling a street scene. In the figure, arrangement faces are numbered from 1 to 41. Here is the list of the level-0 areal objects within the figure: four buildings { 1, 2, 3, 4 }, eight sidewalks { 5, 6, 7, 8, 9, 10, 11, 12 }, four pedestrian crossings { (16, 17, 18), (22, 23, 24), (28, 29, 30), (34, 35, 36) }, twelve lanes { (13, 16), (14, 17), (15, 18), (19, 22), (20, 23), (21, 24), (25, 28), (26, 29), (27, 30), (31, 34), (32, 35), (33, 36) } and an intersection { (37, 38, 39, 40, 41) }. Note that, not all objects can be displayed at the same time due to shared surfaces; parent objects constructed on these level-0 objects can be seen in *Figures* 3.3 and 3.4.

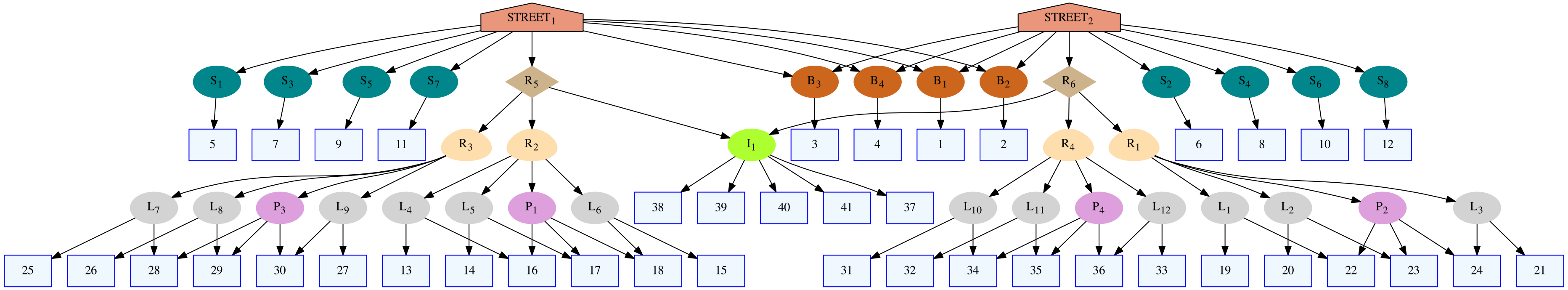


**Figure 3.3:** Representing semantic object hierarchy with a DAG: B → building, I → intersection, L → lane, P → pedestrian crossing, R → road, S → sidewalk. The leaf nodes represent the cells (vertices, edges, faces) of the arrangement modelling the underlying planar partition. Please refer to *Figure* 3.2 for the physical surfaces corresponding to these leaf nodes in the graph.

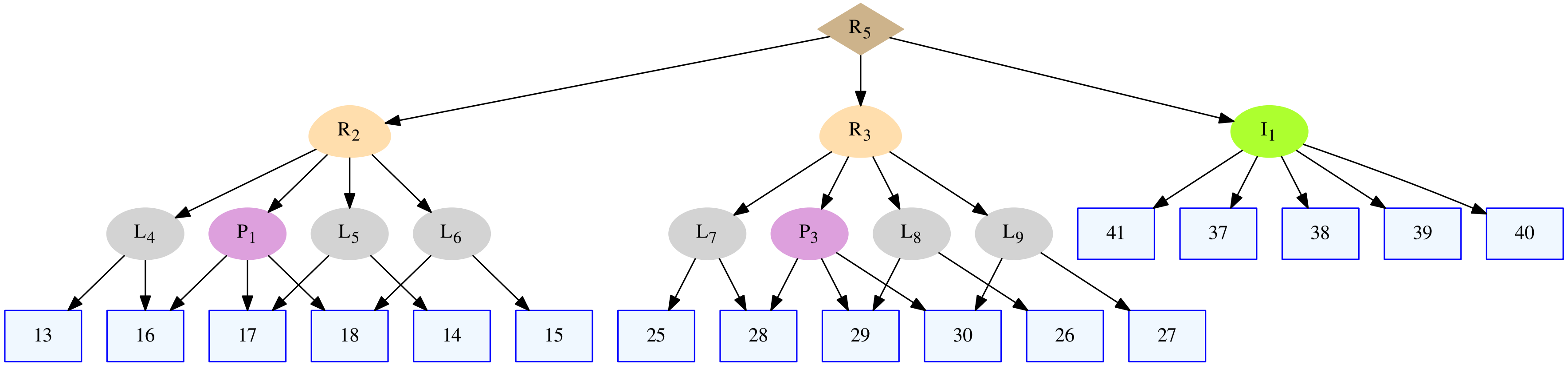


**Figure 3.4:** A sub-graph of the object hierarchy: this graph is a sub-graph of the graph given in *Figure* 3.3. Notice that a sub-graph of a DAG is also a DAG.

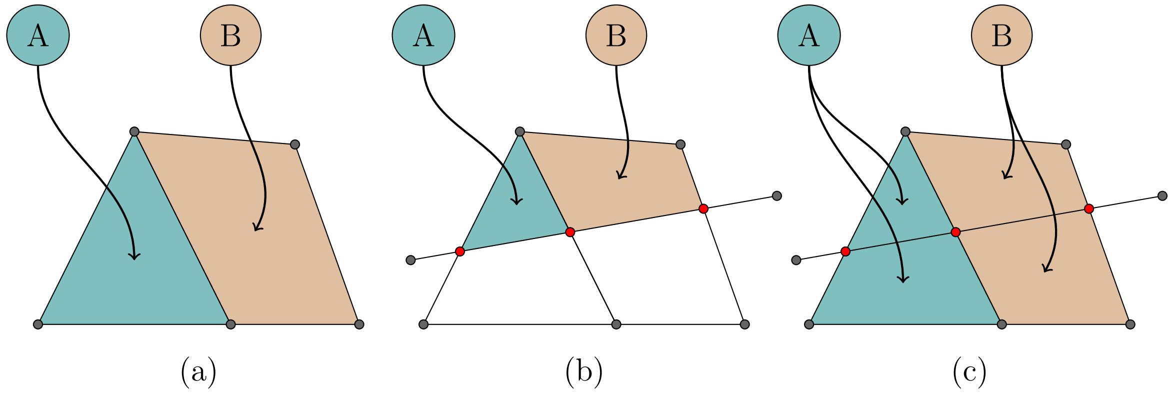


**Figure 3.5:** An example for the maintenance of semantic objects: (a) initial configuration of two objects; (b) a segment is inserted into the arrangement which splits two faces and the two corresponding objects become invalid, i.e., their initial geometry is shrunk; (c) the objects are notified for the split of the faces and the other half of the split faces are linked to the semantic objects. Therefore, the geometry of the already constructed objects are kept unchanged.

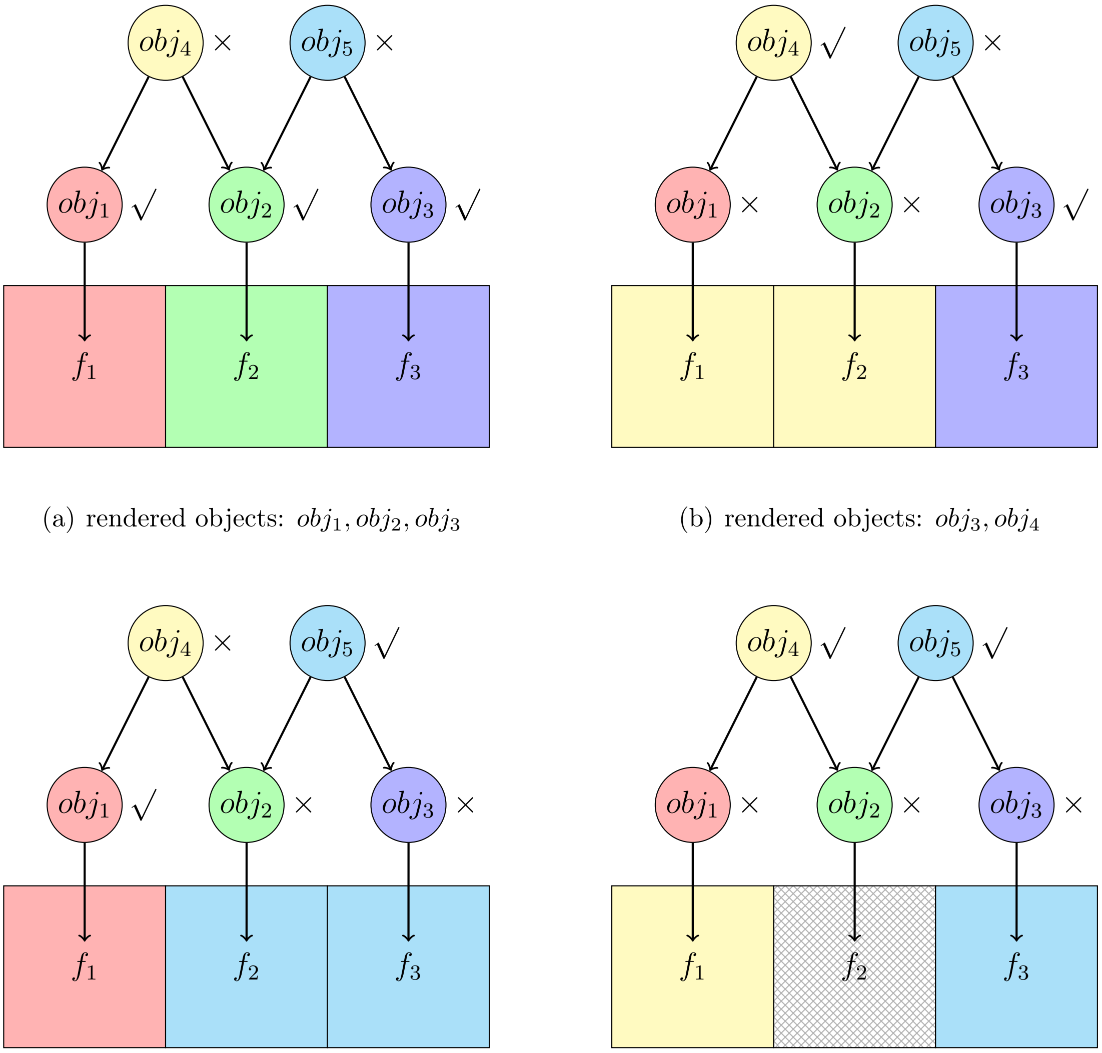
be deleted without violating the validity of a semantic object. In order to remove a linked cell, first the associated objects have to be deleted from the data model. Similarly, deletion of child objects are also constrained such that only the semantic objects that correspond to source nodes in the object hierarchy can be deleted from the data model. After a deletion of a source node in the hierarchy, the children of the source node might become source nodes as well if they do not have other parent objects. In short, only the objects which do not have any parent objects can be deleted from the object model.

## 3.5 Displaying Object Hierarchy on the Computer Screen

Visible semantic objects are rendered on the computer screen with the corresponding predefined style (color, line width, fill pattern, etc.) for each object class. Notice that, the object hierarchy is constructed on the same set of geometric primitives (arrangement cells). Therefore, the rendering system needs to manage the display of the parent-child objects. However, ambiguity conditions may arise when rendering the parent objects sharing the same child objects. That is, shared objects can be the source of ambiguities and the parent objects sharing the same children can be affected from these ambiguities.

An immediate solution to the mentioned rendering problem could be using color blending or transparency. This would be a nice solution for a restricted hierarchy of objects in which an object might have a limited number of parents. However, in our system, there is no such limit and an object can be shared by many other objects in different hierarchical levels. Therefore, using color transparency or blending might be too cumbersome in our system to identify which surface belongs to which object. As a result, a solid rendering problem arises.

*Figure* 3.6 displays an hierarchy of five objects constructed on top of an arrangement composed of 3 faces, 10 edges and 8 vertices. In the figure, objects $obj_1$, $obj_2$ and $obj_3$ are level-0 objects and they are directly linked to the underlying arrangement cells. On the other hand, $obj_4$ and $obj_5$ are level-1 objects and they are constructed on top of the the level-0 objects. *Figure* 3.6 displays four possible rendering configurations a, b, c, d of the given object hierarchy. In each of these configurations, some of the objects are visible whereas some are not. Observe that, there exists an ambiguity condition for the rendering given in *Figure* 3.6(d). In this sub-figure, both parent objects ($obj_4$, $obj_5$) are wanted to be displayed at the same time which is technically impossible since the arrangement face ($f_2$) linked to the shared object ($obj_2$) cannot be rendered with two different settings at the same time.



(a) rendered objects: $obj_1, obj_2, obj_3$

(b) rendered objects: $obj_3, obj_4$

(c) rendered objects: $obj_1, obj_5$

(d) rendered objects: $obj_4, obj_5$

**Figure 3.6:** All possible renderings (a, b, c, d) of a simple object hierarchy: three level-0 objects ($obj_1$, $obj_2$, $obj_3$) and two level-1 objects ($obj_4$, $obj_5$) are constructed on top of an arrangement (faces of the arrangement → $f_1$, $f_2$, $f_3$). Visible (rendered) and non-visible (non-rendered) objects are marked with $\surd$ and ×, respectively. The rendering configuration (d) is problematic since the arrangement face $f_2$ cannot be rendered with the settings of $obj_4$ and $obj_5$ at the same time. Ambiguity conditions arise when parent objects sharing child objects are wanted to be rendered at the same time.

In StreetMaker, users are allowed to change the visibility (rendering) of the individual objects which effectively changes the rendering configuration of the object hierarchies.

That is, the transitions between different rendering configurations are done via changing the visibility of individual objects. Furthermore, StreetMaker has a feature called ambiguity solver which automatically adjusts the visibility of the objects and prevents the ambiguity conditions. For our simple example given in *Figure* 3.6, *Figure* 3.7 shows the transition graphs between different rendering configurations when the ambiguity solver feature is enabled and disabled. An edge in the rendering transition graph corresponds to a change in the visibility of a single object made by the user. In effect, the visibility of many objects are adjusted to make this happen. When the user changes the visibility of an object the ambiguity solver kicks in and searches the object hierarchy graph for the ambiguity conditions. For each detected ambiguity condition, ambiguity involved objects (ambiguity source and ambiguity affected) are identified. Then, by adjusting the visibilities of these objects, the ambiguity conditions are automatically resolved. More information on the resolving of the ambiguity conditions are given in the implementation details (*Section* 3.6).

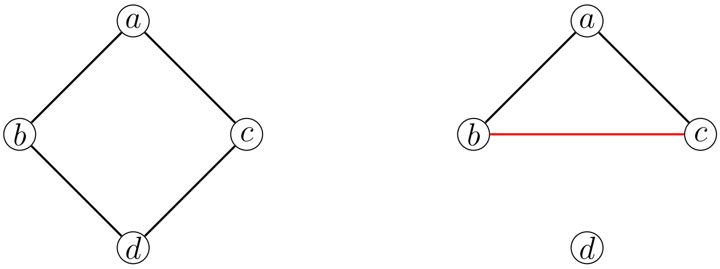


**Figure 3.7:** Rendering transitions for the object hierarchy given in *Figure* 3.6 when the ambiguity solver is disabled (on the left) and enabled (on the right). The graph nodes represent different rendering configurations (a, b, c, d) and each graph edge corresponds to a change in the visibility of a single object initiated by the user. For instance, when the current rendering configuration is (a), making $obj_4$ or $obj_5$ visible transposes the rendering configuration to (b) and (c) respectively. Moreover, if the ambiguity solver is disabled, transitions to configuration (d) is possible from (b) by making $obj_5$ visible or from (c) by making $obj_4$ visible. On the contrary, when the ambiguity solver is enabled, ambiguity conditions are avoided by automatically transforming configuration (b) to (c) or vice versa.

## 3.6 Implementation Details

Semantic objects are stored in a hash table within StreetMaker. Object ids (UUID) [ITU-T X.667, 2012] are used as keys to locate the objects within the hash table. Moreover, the nodes of the object hierarchy graph only store the object ids and by using these ids any object can be retrieved from the hash table.

There exists two methods for creating semantic objects. The first method is fully manual in which the objects are created on the selected arrangement cells. On the other hand, the second method is fully automatic in which the geometry of the semantic objects are imported from an external data source. In addition, possibly existing semantic data in the input data source can be extracted and stored with the created semantic objects. In order to use automatic object importing feature, the user has to define an XML file which maps the related elements in the input data source to automatically created semantic objects. Creating semantic objects are further discussed in *Section* 5.4.

In a rendering cycle, arrangement cells are iterated one by one and rendered with inherited settings from one of the parent objects. Recall that level-0 (leaf) nodes in the hierarchy graph correspond to arrangement cells and the objects that are linked to arrangement cells are of type level-0 (level-1 nodes in the hierarchy graph) and composite. The algorithm behind the ambiguity solver depends on the DAG structure and a local variable called *display level* that is defined for each object directly linked to the arrangement cells, i.e., for level-0 and composite objects. The *display level* of an object is used to conclude on the rendering settings of the corresponding arrangement cells. For instance, the arrangement cells associated to an object whose *display level* equals to 0, are rendered with the object's own rendering settings. On the other hand, for an object with a *display level* different than 0, the corresponding arrangement cells (the cells that are linked to the object) are rendered with the settings of a parent object whose level[6] is equal to the *display level* of the object. *Table* 3.2 displays the *display levels* of the objects and the rendering configurations for the simple example analysed in the previous section (*Figure* 3.6).

| **Rendering Configuration** | **Object Display Level** $(obj_1, obj_2, obj_3)$ | **Face Rendering** $(f_1, f_2, f_3)$ |
|---|---|---|
| (a) | (0, 0, 0) | $(obj_1, obj_2, obj_3)$ |
| (b) | (1, 1, 0) | $(obj_4, obj_4, obj_3)$ |
| (c) | (0, 1, 1) | $(obj_1, obj_5, obj_5)$ |
| (d) | (1, 1, 1) | $(obj_4, \times, obj_5)$ |

**Table 3.2:** Rendering arrangement cells: the rendering of arrangement cells and the display levels of the objects given in *Figure* 3.6. The character × (in the rightmost column of the last row) indicates the ambiguity in the rendering of the face $f_2$.

When the user makes an object visible, generally the rendering of a set of objects are affected. For instance, observe in *Table* 3.2 that from rendering configuration (a) to (b) the *display levels* of $obj_2$ and $obj_3$ are change to 1, which in effect makes the $obj_4$ visible, $obj_1$ and $obj_2$ invisible. For this transition from (a) to (b), the user only changes the visibility of $obj_4$ (by simply right clicking on the object in the object hierarchy window, see *Section* 5.3). Upon user's request for making the $obj_4$ visible, the level-0 children of $obj_4$ are found within the DAG and their *display levels* are set to the level of $obj_4$ which is 1. From rendering configuration (b) to (a), the user has to make a request to make either $obj_1$ or $obj_2$ visible. Let's assume that $obj_1$ is selected to be made visible for this case. $obj_1$ is already a level-0 object and it has no children, therefore its *display level* is set to its level (0) which makes it visible. At this point, $obj_2$'s *display level* remains as 1 which breaks the integrity of the rendering since $f_2$ is continued to be rendered with the settings of $obj_4$ which should not be visible. Therefore, special care has to be taken when a child object is made visible while one of its parent objects has been visible. The integrity of the rendering is maintained by decreasing the *display levels* of the non-updated children under the parent object. The amount of the decrease is set to 1 which also makes the other children of the parent object visible. That is, decreasing the *display level* of $obj_2$ by 1 (*display level* $= 1 - 1 = 0$) fixes the problem and makes it also visible.

Ambiguity conditions are searched through the shared level-0 objects. For instance, when the current rendering configuration is (b) and the user makes the $obj_5$ visible, then the

[6] The level of an object is one less than its level in the hierarchy graph.

*display levels* of $obj_2$ and $obj_3$ are set to 1 (the level of $obj_5$). Observe that the *display level* of $obj_1$ is also 1, i.e., $obj_4$ is also visible and the ambiguity condition arises. When the shared object ($obj_2$) is analysed, it is found out that it has two parents visible at the same time ($obj_4$ and $obj_5$). Ambiguity solver finds out that $obj_5$ is made visible more recently than $obj_4$ and changes the *display levels* of $obj_4$'s level-0 children without interfering with the visibility of the $obj_5$. That is, only the *display level* of $obj_1$ is changed and it is decremented by one. As a result, the ambiguity condition is automatically solved and $obj_1$ and $obj_5$ become visible, i.e. a transition from rendering configuration (b) to (c) is attained.

Ambiguity conditions can also be solved by manually changing the visibility of the objects one by one. However, it becomes tricky and takes more time when the number of the objects increases. Furthermore, ambiguity solver can be disabled if needed. In this case, the ambiguity affected cells of the underlying arrangement are rendered with a distinctive set of settings (e.g. with a special pattern chosen by the user) which indicates the ambiguity conditions visually to the users.

## 3.7 Conclusion

To sum up, we presented a DAG-structured object hierarchy constructed on top of a planar partition represented by a 2D arrangement. In this hierarchy, the graph nodes represent either the semantic objects or the underlying geometric primitives / arrangement cells. The nodes that correspond to arrangement cells are level-0 (leaf nodes) and the rest of the nodes correspond to semantic objects at different levels. An edge between two semantic object nodes indicates a thematic parent-child relationship between the objects. On the other hand, an edge between a level-0 node and a non-level-0 node represents a link between the geometry (an arrangement cell) and the corresponding semantic object.

The ambiguity conditions related to the rendering of semantic objects and our solution to this problem has been discussed. In order to manage the rendering of the objects we utilized the DAG-structured hierarchy together with a set of local variables (*display levels*) attached to level-0 and composite objects. To the author's knowledge, this kind of approach is used for the first time for solving such a rendering problem.

Maintenance of planar partitions are discussed in *Chapter* 2 and in this chapter we further raised the maintenance issues to semantic object level. Object hierarchies similar to ours can be constructed using the PostGIS Topology package[7] (Topogeometry objects in PostGIS). However, to the author's knowledge, the maintenance is not performed automatically in PostGIS Topology on the contrary to ours.

Bertin [1967] published his foundational work (originally in French) in the design and cartography which has been affecting the information visualization theory. As a future work, the rendering of the semantic objects might be reconsidered along the graphical techniques (shape, orientation, color, texture, volume and size) as described by Bertin.

[7] http://postgis.net/docs/manual-dev/Topology.html

# Chapter 4

# Generic Algorithms

GISs use variety of algorithms that were developed within the fields of computational geometry, graph theory and computer science. In this chapter, a set of these algorithms and their theory is presented upto a limit. These algorithms are quite generic meaning that they can be used within the context of many GISs. Therefore, having a set of these algorithms that are ready to run on a given dataset would be very beneficial. In fact, we will see in *Chapter* 5 that the algorithms discussed here were implemented as a part of our GIS framework.

## 4.1 Delaunay Triangulations and Voronoi Diagrams

### 4.1.1 Introduction and Definitions

Triangles are very suitable building blocks for space partitioning since a triangle is the simplest planar structure to represent a patch of surface in 2D/3D space. In general, a triangulation can be described as the subdivision of a plane into triangular regions except the unbounded region. The 3D counterpart of a triangulation is the tetrahedralization which should not be confused with a triangular mesh. A triangular mesh represents a 2-manifold surface embedded in 3D but it is neither a triangulation nor a tetrahedralization. For instance, a Digital Elevation Model (DEM) is generally represented by a triangular mesh (Triangulated Irregular Network (TIN)) which is constructed by first triangulating a set of planar points and then lifting the points to 3D by imposing height information on the points [van Kreveld, 1997].

Triangulation of a plane is a very broad subject on its own. Below, some definitions are given in order to provide a more formal definition of triangulations and alpha shapes which is explained in *Section* 4.2. Please refer to Hjelle and Dæhlen [2006]; de Loera et al. [2010]; Devadoss and O'Rourke [2011] for more aspects of triangulations that are not discussed within the scope of this dissertation.

Let $S$ be a finite set in $d$-dimensional Euclidean space ($R^d$) with $k+1$ elements: $S = \{p_0, p_1, ..., p_k\}$. Unless otherwise is stated, $S$ is assumed to be this set in the following definitions.

***affine combination and affine hull:*** An affine combination of the points $p_i \in S$ is a point $x = \sum \alpha_i p_i$ with $\sum \alpha_i = 1$ and the affine hull of $S$ is defined as the set of all affine combinations of $p_i \in S$. Geometrically, the affine hull of $S$ is the intersection of the all hyperplanes that contain $S$. For instance, affine hull of a single point is the point itself, affine hull of two points is the line passing through the two points and affine hull of three points is the plane defined by the three points. Furthermore, the points $p_i \in S$ are *affinely independent* if none is the affine combination of the others. Consequently, $k+1$ points are affinely independent if and only if the $k$ vectors $v_i = p_i - p_0$, for $1 \leq i \leq k$, are linearly independent. One can find at most $d$ linearly independent vectors in $R^d$, and therefore at most $d+1$ affinely independent points.

***convex set:*** $S$ is a convex set if any line segment joining two points in $S$ lies entirely in $S$ (*Figure* 4.1).

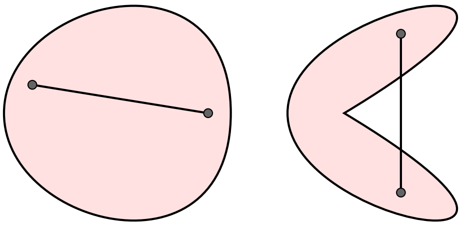


**Figure 4.1:** A convex (left) and a non-convex set (right)

***convex combination and convex hull:*** A convex combination is an affine combination with non-negative coefficients. That is, a convex combination of the points $p_i \in S$ is a point $x = \sum \alpha_i p_i$ with $\sum \alpha_i = 1$ and $\alpha_i > 0$. Then, the convex hull of $S$ (conv($S$)) is defined as the set of all convex combinations of $p_i \in S$. Note that, convex hull of $S$ is equivalent to the smallest convex set containing $S$ (*Figure* 4.2).

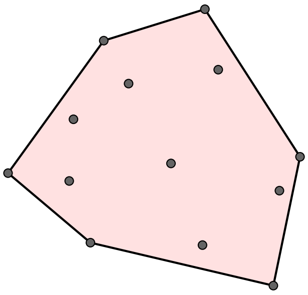


**Figure 4.2:** Convex hull of a point set

***k-simplex:*** A k-simplex, denoted by $\sigma$, is the convex hull of $k+1$ affinely independent points, $\sigma = conv(S = \{p_0, p_1, ..., p_k\})$. $k$ represents the simplex dimension. Remember that one can find at most $d+1$ affinely independent points in $R^d$. As a result, there exists four simplices in $R^2$ and five simplices in $R^3$ including the empty set which is denoted by $(-1)$-simplex. *Figure* 4.3 displays all possible simplices in $R^3$ other than the $(-1)$-simplex.

Any subset of an affinely independent set is also affinely independent and therefore also defines a simplex. A face $\tau$ of $\sigma$ is the convex hull of a subset of $S$, i.e., $\tau = conv(T)$ where $T \subseteq S$. $\tau$ is called a proper face if $T \neq \emptyset$ and $S - T \neq \emptyset$, otherwise it is called an improper face, i.e., $\tau$ is improper if $T = \emptyset$ or $T = S$.

***simplicial complex:*** A simplicial complex $K$ is a finite collection of simplices such that

i. if $\tau$ is a face of $\sigma \in K$, then $\tau \in K$.

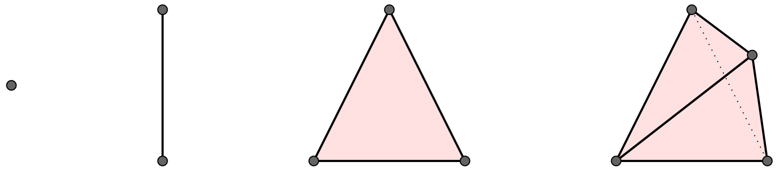


**Figure 4.3:** Left to right: 0-simplex (vertex), 1-simplex (edge), 2-simplex (triangle) and 3-simplex (tetrahedron)

ii. if $\sigma_1, \sigma_2 \in K$, then $\sigma_1 \cap \sigma_2$ is a face of both $\sigma_1$ and $\sigma_2$.

*Figure* 4.4 displays three finite collections of simplices which do not form a simplicial complex. The dimension of $K$ is the largest dimension of any simplex in $K$, i.e., $dim(K) = max\{\, dim(\sigma) \,|\, \sigma \in K \,\}$. A simplicial complex with dimension $k$ is called a k-complex and it is pure if every simplex is a face of a k-simplex. In addition any subset $K'$ of $K$ is a subcomplex of $K$ if $K'$ is also a simplicial complex.

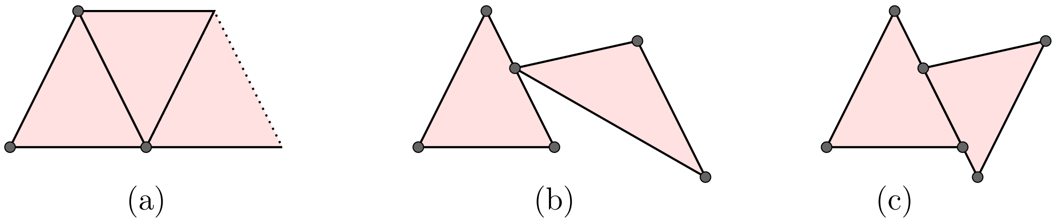


**Figure 4.4:** Examples of simplex collections which do not form simplicial complexes: (a) missing two vertices and an edge hence violating the first requirement; (b), (c) intersection of two triangles is not a face of the triangles hence violating the second requirement.

***triangulation:*** A triangulation $T$ of a point set $P \in R^2$ is a 2-complex $K$, such that all 0-simplices (vertices) of $K$ are points in $P$ and the union of all simplices in $K$ is equal to the convex hull of $P$ ($conv(P)$).

In a triangulation if an internal edge is a diagonal of a convex quadrilateral spanned by the two adjacent triangles, it can be changed with the other possible diagonal within the quadrilateral which results in a different triangulation (*Figure* 4.5). As a result, a new triangulation of the same set is obtained different than the previous one. This observation emerges two questions: how many different triangulations are there for a given arbitrary point set and is it possible to convert a given triangulation to any other triangulation of the same point set by just flipping edges? The first question has not been fully answered yet. Although, there exist formulas for some special cases, a formula for a general case has not been found. For instance, a convex polygon with n-vertices ($n \geq 3$) is one of the special cases. In this case, the number of possible triangulations $T_n$ is given by the following formula [de Loera et al., 2010]:

$$T_n = \frac{1}{n-1}\binom{2n-4}{n-2} \tag{1}$$

Observe that the number of triangulations increase exponentially as the number of points on the convex polygon increase $(1, 2, 5, 21, 42, 131, ...,)$. There is no formula for the general case, but there is an algorithm to find the number of different triangulations for an arbitrary point set given by Avis and Fukuda [1996]. In addition, Aichholzer et al. [2004] and Sharir and Welzl [2006] discovered a lower and an upper asymptotic bound respectively.

$$\text{lower bound: } \mathcal{O}(2.33^n) \quad \text{upper bound: } \mathcal{O}(\frac{59^v 7^b}{\binom{v+b+6}{6}})$$

where $b$ and $v$ are the numbers of boundary and interior vertices respectively and $n$ is the total number of vertices ($n = b + v$).

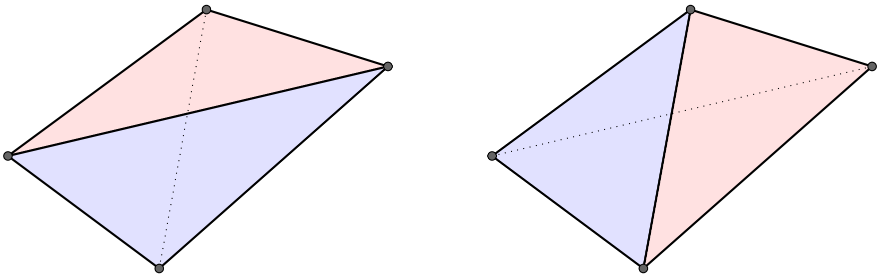


**Figure 4.5:** Edge flip operation: flipping the diagonal of a quadrangle formed by two adjacent triangles

The second question is related to the connectivity of the flip graph of a point set. The nodes of the flip graph are composed of all possible triangulations of the point set and the ones that can be convertible to each other by a single edge flip are connected. Lawson [1972, 1977] proved that flip graph of any point set is connected. That is, any two different triangulations of a point set can be connected to one another through a finite sequence of flips. Another conclusion that can be drawn from this theorem is, the number of triangles for every triangulation of a point set is same since an edge flip, does not change the number of triangles. Observe *Figures* 4.6 and 4.7 for all possible triangulations of a simple point set and its flip graph.

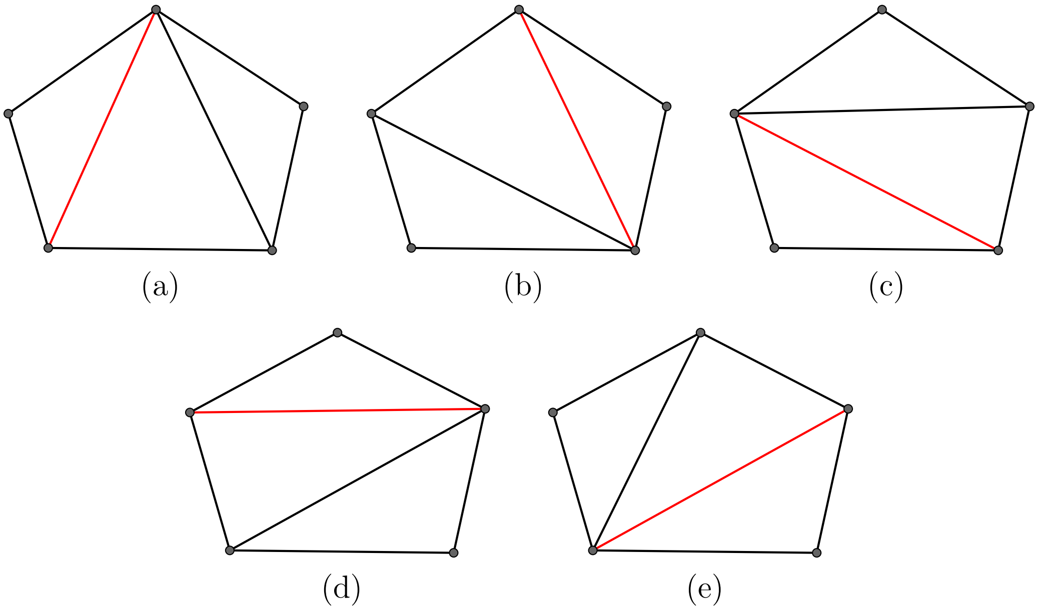


**Figure 4.6:** Five possible triangulations of a convex polygon composed of five points. Each consequential triangulation is obtained via flipping the red edge in the previous triangulation.

We have seen that triangulation of point sets are not unique; however, among them some of the triangulations are preferable to others for practical reasons. In many applications, triangulations which have skinny triangles are not generally favoured. On the contrary,

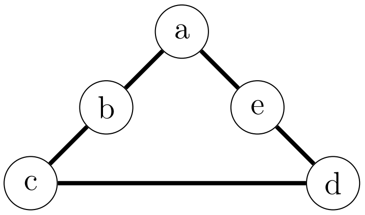


**Figure 4.7:** Flip graph of the simple polygon displayed in *Figure* 4.6. Node labels correspond to figure labels in *Figure* 4.6

triangles close to equilateral triangles are preferred. Delaunay triangulations are optimal triangulations in terms of triangle shapes. There are myriad of applications of Delaunay triangulations and their dual Voronoi diagrams, such that there is a conference series called International Symposium on Voronoi Diagrams in Science and Engineering (ISVD) which only focuses on these structures and their applications, e.g. in pattern recognition, robot motion planning, cartography and crystallography to name a few. Moreover, in GIScience, Voroni diagrams are among the traditional spatial data models and similarly, Delaunay triangulations are mainly used for terrain modelling especially in the TIN construction.

### 4.1.2 Delaunay Triangulations and Voronoi Diagrams

The 'shape' of a triangle can be measured by the triangle's minimum interior angle. For an equilateral triangle each interior angle is 60° and for an arbitrary triangle, the more the minimum interior angle deviates from 60°, the more the triangle becomes skinny. Consequently, the minimum interior angle of a triangle can be used to measure its 'distance' to an equilateral triangle. This observation can be used to compare different triangulations of a point set in terms of the triangle shapes. Assume for a finite point set $P$, there exist $k$ different triangulations each composed of $n$ triangles. One can construct a non-decreasing vectors of minimum angles $V(T_i)$ for every triangulation such that each entry in the vectors corresponds to the minimum angle of a triangle in $T_i$. In mathematical notation:

$$V(T_i) = \{\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n\} \ for \ 1 \leq i \leq k, \ and \ \alpha_s \leq \alpha_t \ for \ s < t, \ and \ s, t \leq n$$

A lexicographic order of vectors $V(T_i)$ gives the order of triangulations in terms of optimality defined above. Among these triangulations the one with the largest lexicographic order is called *Delaunay triangulation* which is named after a Russian mathematician Boris Nikolaevich Delone who first defined the notion [Delaunay, 1934].

Observe that one can construct a Delaunay triangulation from an arbitrary given triangulation by continuously performing edge flip operations until a triangulation with the largest lexicographical order is reached. Note that, an edge flip changes the values of six angles within a triangulation, hence the lexicographical order of the triangulation's minimum angle vector also changes. Edges whose flip increase the order are called *illegal* while the ones whose flip decrease the order are called *legal* edges. When all illegal edges are flipped iteratively, a Delaunay triangulation is obtained.

Delaunay triangulations are unique if all the points are in general position; that is, no three points are collinear and no four or more points are co-circular. For instance, in

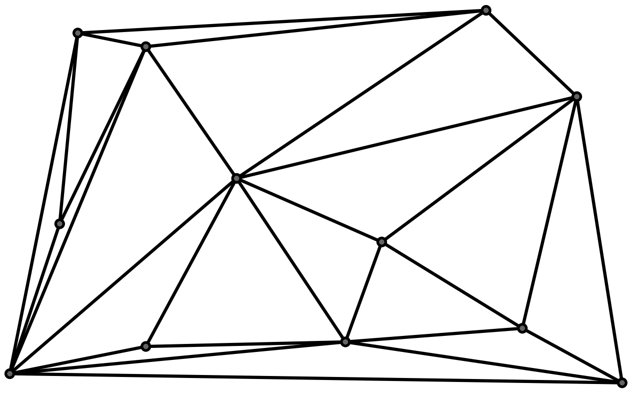


**Figure 4.8:** A triangulation of a point set in $R^2$

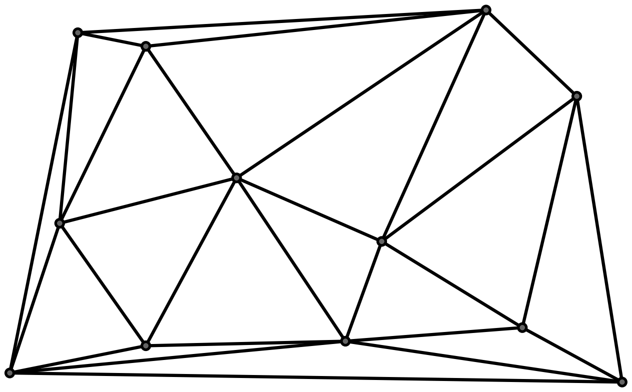


**Figure 4.9:** Delaunay triangulation of the same point set displayed in *Figure* 4.8

the case of four co-circular points that form a quadrangle, there exist two possible triangulations depending on which diagonal of the quadrangle is chosen as an edge in the triangulation. Both triangulations may be optimum in terms of maximum of the minimum angles criterion. For instance, the triangulations of a rectangle yield exactly the same angles. Please refer to Hansford [1990] for a detailed analysis of Delaunay triangulations in the existence of four or more co-circular points. From now on, unless otherwise is stated, the given points are assumed to be in general position.

The definitions of legal and illegal edges given above can be reformulated using well known theorems on triangles and circles since ancient Greeks. Let $\Diamond ABCD$ be a convex quadrangle triangulated by the triangles $\triangle ABC$ and $\triangle ACD$. The edge $\overline{AC}$ is a legal edge if the vertex $D$ is outside the circumcircle of $\triangle ABC$ and it is illegal if it is inside the circumcircle (*Figure* 4.10). Please see *Appendix*-B.1 for a proof. As a result of this reformulation, we can give another equivalent definition for Delaunay triangulations:

Let $T$ be a triangulation of a point set $S$ in general position. $T$ is a Delaunay triangulation if and only if no point from $S$ is in the interior of any circumcircle of a triangle of $T$. In other words, all the edges of a Delaunay triangulation are valid.

A Delaunay triangulation is dual to a *Voronoi diagram* which is another geometric struc-

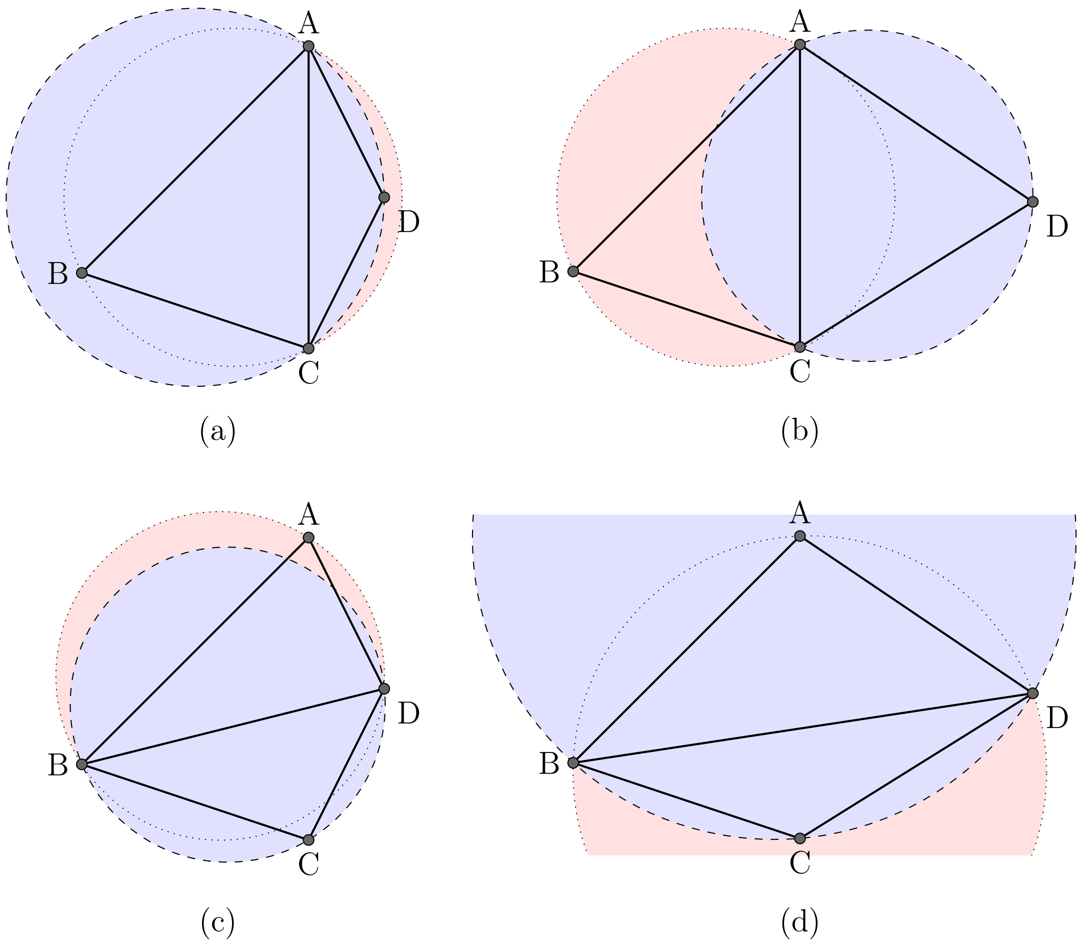


**Figure 4.10:** Legal and illegal edges: $\overline{AC}$ is legal in (b) and illegal in (a); $\overline{BD}$ is legal in (c) and illegal in (d). Observe that (c) is generated from (a) and (d) is generated from (b) by flipping the edge $\overline{AC}$. A flip of a legal / an illegal edge results in an illegal / a legal edge.

ture based on the Euclidean distance, named after Georges Voronoï [Voronoï, 1907, 1908], also known as Thiessen polygons and Dirichlet tessellations [Aurenhammer and Klein, 2000]. The formal definition of Voronoi diagrams is as follows:

Let $S = \{p_1, p_2, ..., p_N\}$ be a set of distinct points in $R^2$ and let $d(i, j)$ denotes the Euclidean distance between $p_i$ and $p_j$. The elements of $S$ are called sites within the context of Voronoi diagrams. A region, called Voronoi region, is associated to each site in the plane, denoted by $vor(p_i)$, such that

$$vor(p_i) = \{x : d(x, p_i) \leq d(x, p_j)\}, \quad x \in R^2, \quad j = 1, 2, 3, ..., N$$

That is, $vor(p_i)$ is composed of all points within the plane that are at least as close to $p_i$ as to any other site in $S$. The points that lie on the boundary between regions do not have a unique nearest site. The Voronoi diagram of $S$, denoted by $Vor(S)$, is defined by the union of all the boundary points between the regions (*Figure* 4.11).

If $S$ is composed of only two sites, then $Vor(S)$ is the perpendicular bisector of the line segment $p_1p_2$. This bisector cuts the plane into two half planes, in which the two sites lie. Thus, for this case $vor(p_1)$ and $vor(p_2)$ are half planes denoted by $H(p_1, p_2)$ and $H(p_2, p_1)$ respectively. When $S$ has more than two sites, we can continue to draw perpendicular

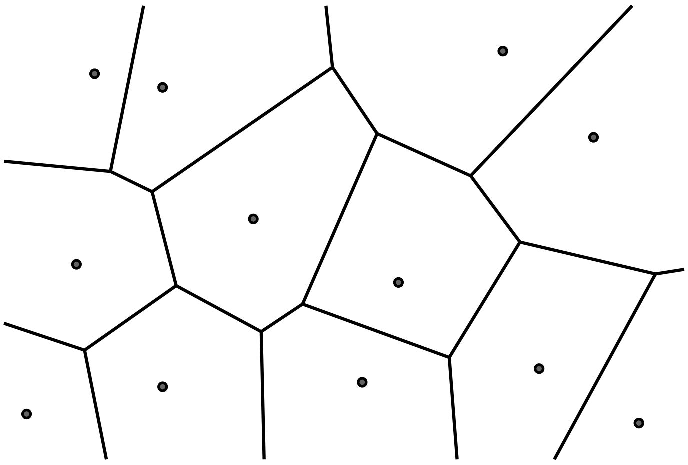


**Figure 4.11:** Voronoi diagram of a point set in $R^2$

bisectors of segments joining two different sites, each of which cuts the plane into two half planes. As a result, we can define the Voronoi region of a site $p_i$ as the intersection of all half planes $H(p_i, p_j)$ where $p_j$ is any other site in $S$. Then, the vertices of a Voronoi diagram, called Voronoi vertices, are a subset of the intersection of the perpendicular bisectors and similarly, the edges of a Voronoi diagram, called Voronoi edges, are a subset of the perpendicular bisector segments and rays. Furthermore, half planes are convex sets and intersections of convex sets are also convex; hence all Voronoi regions are convex (*Figure* 4.12).

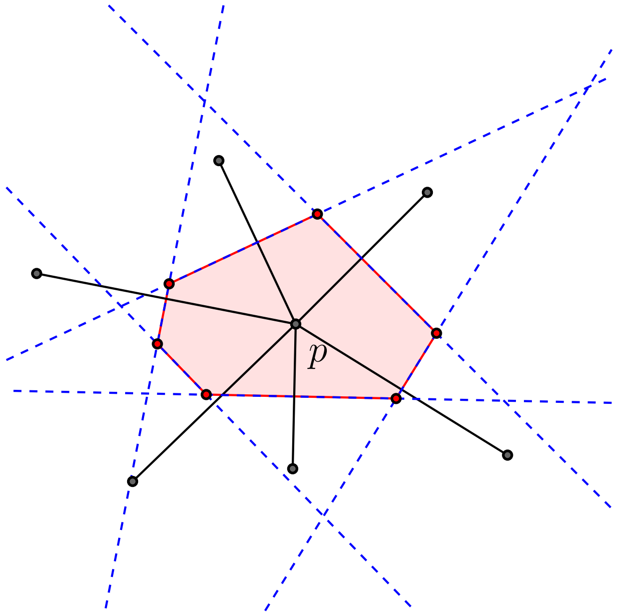


**Figure 4.12:** Voronoi region of a site $p$ is defined by the intersection of half planes: black solid lines display the segments joining the sites; blue dashed lines are the perpendicular bisectors to the segments; the shaded region is $vor(p)$ whose boundary is defined by the intersections of the bisectors.

For a Voronoi diagram of at least three non-collinear sites, all Voronoi vertices are degree three; i.e., they are on the border of three Voronoi regions or equivalently they are connected to three Voronoi segments or rays. This statement is valid when no four or more points are co-circular, i.e., when all points are in general positions. For instance, in the case of four co-circular points, a Voronoi vertex of degree 4 is generated (*Figure* 4.13). A theorem from these observations can be constructed which states that: A point $v$ is a Voronoi vertex of $Vor(S)$ if and only if there exists a circle centred at v with three or more sites on its boundary and none in its interior [Devadoss and O'Rourke, 2011]. This theorem leads to the duality principle between Voronoi diagrams and Delaunay triangulations.

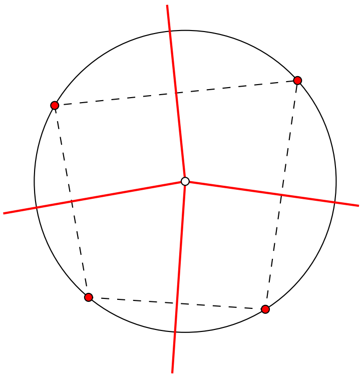


**Figure 4.13:** Voronoi diagram of four co-circular points

Duality principle indicates a bijective mapping (when all points are in general position) from Voronoi diagrams to Delaunay triangulations such that each of the two can be constructed from the other. In this bijection, a Voronoi vertex corresponds to a Delaunay triangle, a Voronoi edge corresponds to a Delaunay edge and a Voronoi region (or a site in Voronoi diagram) corresponds to a Delaunay vertex (*Figure* 4.14). Interested reader may refer to Aurenhammer and Klein [2000] and Devadoss and O'Rourke [2011] for the theorems and their proofs of the duality principle.

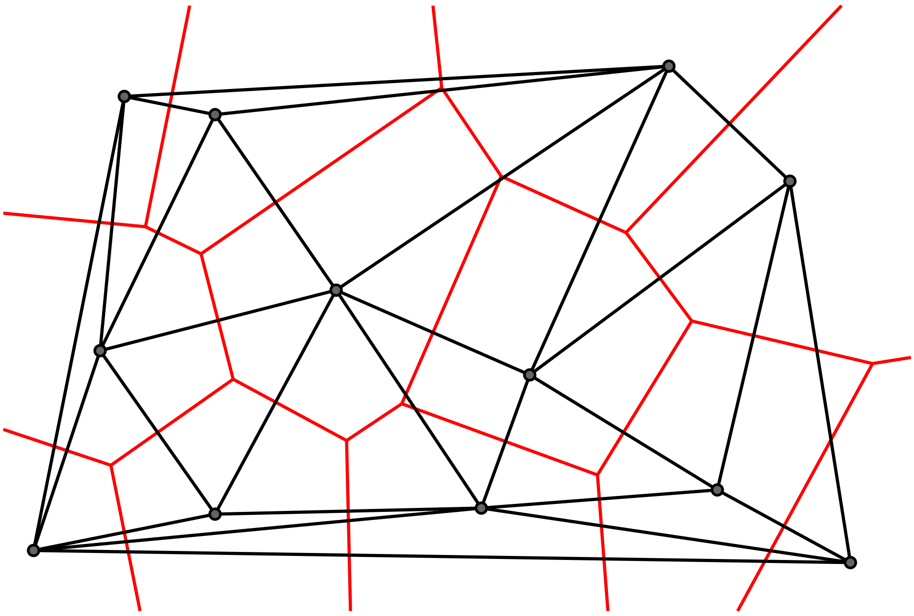


**Figure 4.14:** Duality of Voronoi diagrams and Delaunay triangulations

The concept of Voronoi diagram and Delaunay triangulation described above can be generalized or specialized. For instance, they can be defined for other geometric constructions other than points. A Voronoi diagram whose sites are composed of line segments and points is called Segment Voronoi Diagram (SVD) and its dual is called Segment Delaunay Graph (SDG) [Karavelas, 2004; Alt et al., 2005] (*Figure* 4.15). Another example of a specialization is the constrained Delaunay triangulation in which a set of predefined edges are forced to be part of the triangulation [Joe and Wang, 1993]. And as a final example, weighted Voronoi diagrams can be constructed by using a non-Euclidean distance function.

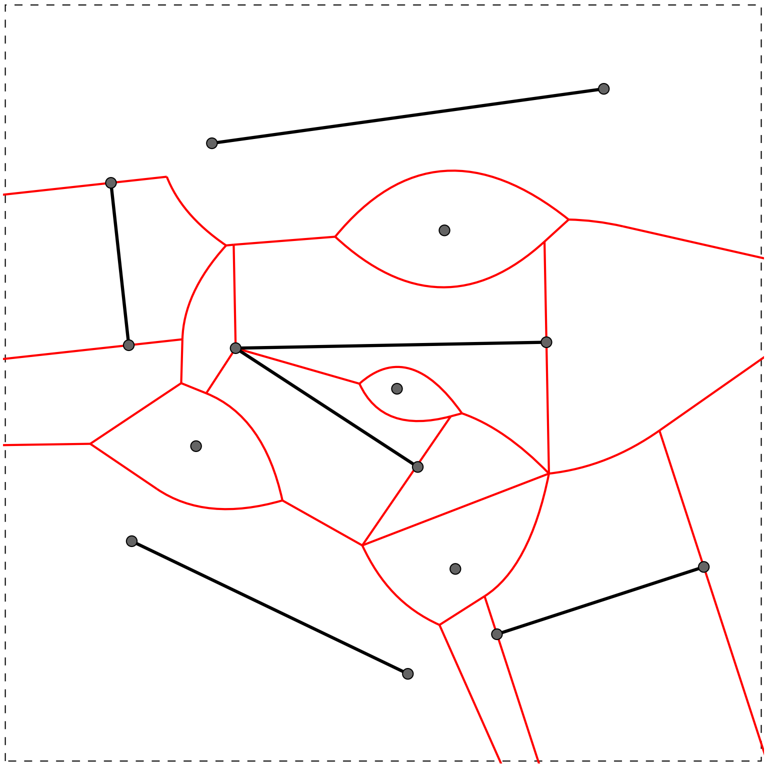


**Figure 4.15:** Segment Voronoi diagram

There exists several algorithms for constructing both Delaunay triangulations and Voronoi diagrams. One can construct these structures using their definitions. According to this direct approach, the Delaunay triangulation of a point set can be obtained by first constructing a random triangulation of the point set and then flipping all the illegal edges repeatedly. Similarly, the Voronoi diagram of the point set can be constructed by intersecting the half planes generated by the perpendicular bisectors between the sites. However, these direct approaches do not lead to optimal algorithms. Moreover, thanks to the duality of these structures, one of them can obtained from the other.

The algorithms for computation of Delaunay triangulations and Voronoi diagrams can be grouped into categories according to their approaches. In this dissertation only two of these groups are mentioned. Please refer to Su and Drysdale [1997] and Liu and Snoeyink [2005] for a comparative survey and classification of the existing algorithms in 2D and 3D respectively. The first group of algorithms are called incremental algorithms [Lee and Schachter, 1980; Watson, 1981; Guibas and Stolfi, 1985]. Incremental algorithms start with an initial triangulation which can be a single triangle. Then the remaining points of the set are inserted to the initial triangulation one by one. After each insertion, the triangulation is updated to be a Delaunay. The worst case runtime of incremental algorithms are quadratic ($\mathcal{O}(n^2)$). The fastest algorithms are grouped into divide and conquer category which are $\mathcal{O}(nlogn)$. The basic idea is to recursively subdivide the input point set into two data sets of approximately equal size until each data set contains only a small number of points. Delaunay triangulations that are constructed at the lowest levels merged recursively to obtain the Delaunay triangulation for the all set. Lee and Schachter [1980] proposed the first divide and conquer algorithm which was based on adjacency list data structure. Later, Guibas and Stolfi [1985] invented the edge algebra and improved the algorithm a bit further by using the quad-edge data structure, and finally Dwyer [1987] improved the initial subdivision step of the algorithm. For the interested reader, here are some other notable papers which follow different approaches than already mentioned algorithms: McLain [1976]; Mirante and Weingarten [1982]; Fortune [1987].

The implementation of Delaunay triangulations and Voronoi diagrams within the context of this work are explained in *Section* 5.6. In addition, the following two topics alpha shapes (*Section* 4.2) and medial axes (*Section* 4.3.1) are directly related to Delaunay triangulations and Voronoi diagrams.

## 4.2 Alpha Shapes

The notion of alpha shapes was first introduced by Edelsbrunner et al. [1983] in order to define the geometric shape of a point set in $R^2$ and later Edelsbrunner and Mücke [1994] established the alpha shapes theory for the point sets in $R^3$ which can further be generalized to $R^d$. Note that the shape of a point set is a vague notion and $\alpha$-shape representation is just one of the many possible interpretations. In the context of this dissertation we are only interested in 2D $\alpha$-shapes, therefore the concepts discussed here are confined to 2D, although they can be easily generalized to 3D.

Assume that $S$ is a finite set of points $p_i \in R^2$ in general position. General position of points implies that no three points in $S$ are collinear and no four points in $S$ are co-circular. These assumptions avoid the complications of the definitions given below. Please refer to Edelsbrunner and Mücke [1994] for dealing with the degenerate cases, i.e., when the points are not in general position. Moreover, in *Figures* 4.16, 4.17, 4.18, 4.19, 4.20, 4.21, and 4.22 the same point set is used and the value of $\alpha$ is fixed.

**$\alpha$-*circle***: A circle with radius $\alpha$.

**$\alpha$-*disk***: An open disk bounded by an $\alpha$-circle, for $0 < \alpha < \infty$. An $\alpha$-disk $d$ is empty if $d \cap S = \emptyset$.

**$\alpha$-*exposed k-simplex***: Let $T \subseteq S$ with $|T| = k + 1$ and $0 \leq k \leq 2$. Also assume that $\sigma_T$ is the k-simplex representing the $conv(T)$. $\sigma_T$ is called $\alpha$-exposed if there exists an empty $\alpha$-disk $d$ such that $T = \partial d \cap S$, where $\partial d$ is $\alpha$-circle. Note that for $|T| = 3$, $\sigma_T$ is a triangle and a triangle can be $\alpha$-exposed if and only if $\alpha$ is equal to the radius of the triangle's circumcircle and the corresponding $\alpha$-disk is empty. On the other hand, for $|T| = 2$, $\sigma_T$ is an edge and two circles can be drawn passing through these points for any value of $\alpha$. That is, $\sigma_T$ is $\alpha$-exposed if at least one of the disks bounded by these circles is empty. Finally, for $|T| = 1$, $\sigma_T$ is a vertex and it is possible to draw infinitely many $\alpha$-disks passing through the vertex some of which might be empty (*Figure* 4.16).

**$\alpha$-*shape***: Let's define two sets for a fixed $\alpha$: the set of $\alpha$-exposed 0-simplices (vertices) denoted by $F_{0,\alpha}$ and the set of $\alpha$-exposed 1-simplices (edges) denoted by $F_{1,\alpha}$. Then, the $\alpha$-

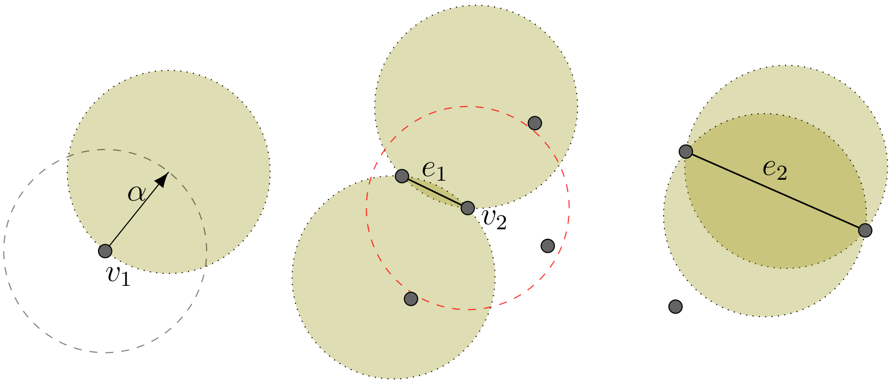


**Figure 4.16:** $\alpha$-exposed $k$-simplices for a fixed $\alpha$: $v_1$ is $\alpha$-exposed since infinitely many empty $\alpha$-disks can be drawn passing through $v_1$ (only one of them is displayed). Note that centres of the all $\alpha$-circles passing through $v_1$ is on the $\alpha$-circle centred at $v_1$. On the other hand, $v_2$ is not $\alpha$-exposed because none of the (infinitely many) $\alpha$-disks passing through $v_2$ is empty. $e_1$ is also not $\alpha$-exposed since corresponding two $\alpha$-disks are not empty. On the contrary, $e_2$ is $\alpha$-exposed. Although only one empty $\alpha$-disk is enough to be $\alpha$-exposed, both $\alpha$-disks are empty for $e_2$.

shape of $S$, denoted by $S_\alpha$, is the polytope whose boundary ($\partial S_\alpha$) consists of edges in $F_{1,\alpha}$ and the vertices in $F_{0,\alpha}$. In algebraic topology, a polytope is defined as the underlying space of a simplicial complex. This simplicial complex is called $\alpha$-complex within the context of $\alpha$-shapes. A more detailed definition of $\alpha$-complex will be given shortly. The interior of $S_\alpha$ is bounded by the edges of $S_\alpha$ that are $\alpha$-exposed only from one direction, i.e., only one of the two corresponding $\alpha$-disks is empty (*Figure* 4.17).

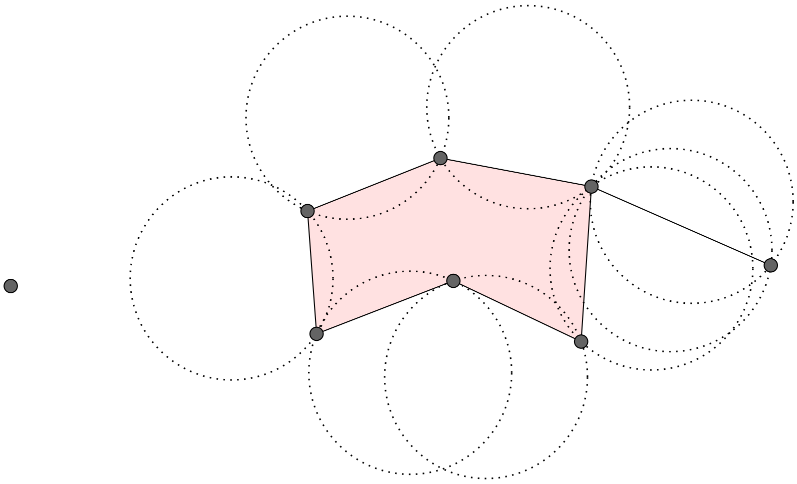


**Figure 4.17:** $\alpha$-shape of the point set: the dotted $\alpha$-circles are certificates for the membership of the solid edges to the $\alpha$-shape. All vertices are also in the $\alpha$-shape and the interior of the $\alpha$-shape is shaded in the figure.

Observe that for a sufficiently small $\alpha$, the set $F_{1,\alpha}$ will be empty and this results in $S_\alpha = S$. In addition for a sufficiently large $\alpha$, $F_{1,\alpha}$ will contain only the edges from *conv*(*S*). That is,

$$\begin{aligned} \lim_{\alpha \to 0} S_\alpha &= S \\ \lim_{\alpha \to \infty} S_\alpha &= conv(S) \end{aligned} \tag{2}$$

Let's denote the Delaunay triangulation of $S$ by $DT(S)$. Recall from *Section* 4.1 that $DT(S)$ is a simplicial complex such that the circumcircle of each 2-simplex (triangle) in $DT(S)$ does not contain any other point of $S$. Thus, we can conclude that any 2-simplex in DT(S) with circumcircle radius $\rho$ is $\alpha$-exposed for $\alpha = \rho$. Furthermore, one can find an appropriate $\alpha$ value for each proper face of a 2-simplex in $DT(S)$ that makes it $\alpha$-exposed. This implies the following statement: For each simplex $\sigma \in DT(S)$, there exist values of $\alpha > 0$ so that $\sigma$ becomes $\alpha$-exposed. Conversely, every face of $S_\alpha$ is a simplex of $DT(S)$ (*Figure* 4.18).

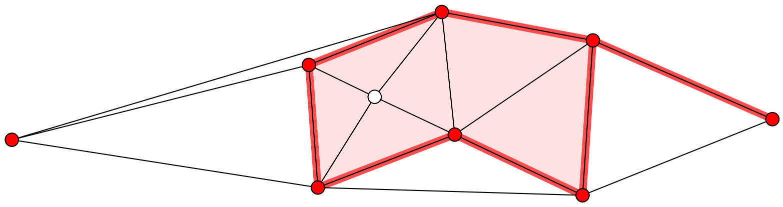


**Figure 4.18:** $\alpha$-shape of a point set is superimposed on its Delaunay triangulation. Notice that, all vertices and edges of the $\alpha$-shape are also simplices of the Delaunay triangulation.

***$\alpha$-hull:*** The $\alpha$-hull of $S$, denoted by $H_\alpha$, is defined as the complement of the union of all empty $\alpha$-disks (*Figure* 4.19).

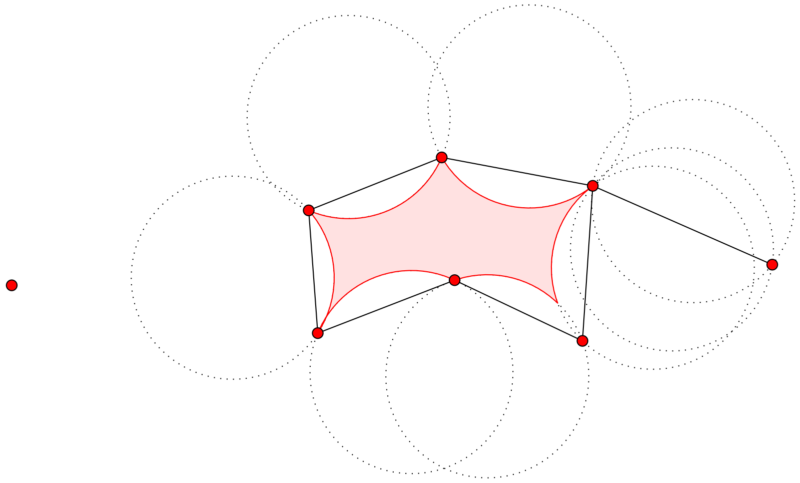


**Figure 4.19:** $\alpha$-hull of the point set: the solid lines represent the border of the $\alpha$-shape and the dotted circular arcs indicate the empty $\alpha$-disks that intersect with the alpha-shape. Note that, all of the vertices are also included in the $\alpha$-hull.

***$\alpha$-diagram:*** The union of all $\alpha$-disks centred at the points $p \in S$ is called the $\alpha$-diagram of $S$ and it is denoted by $U_\alpha$ (*Figure* 4.20). Observe that a point $x \in R^2$ belongs to $U_\alpha$ if and only if the $\alpha$-disk $d_x$ centred at x is not empty. This observation leads to the following close relationship between $U_\alpha$ and $H_\alpha$ (*Figure* 4.21).

$$\begin{aligned} x \in U_\alpha \quad &\Leftrightarrow \quad d_x \cap H_\alpha \neq \emptyset \\ x \in H_\alpha \quad &\Leftrightarrow \quad d_x \subseteq U_\alpha \end{aligned} \tag{3}$$

The boundary of $U_\alpha$, denoted by $\partial U_\alpha$, consists of circular arcs that meet at corner points. Assume that $T = \{p_1, p_2\}$ with $T \subseteq S$ and $\sigma_T$ is $\alpha$-exposed due to an $\alpha$-disk $d$. Then, observe the fact that the centre of $d$ has to be on one of the intersection points of the two $\alpha$-circles centred at $p_1$ and $p_2$. This shows a correspondence between the $\alpha$-exposed edges of $S_\alpha$ and the corners of $\partial U_\alpha$. Note that, this correspondence is not a bijection unless we take the multiplicities of edges of $S_\alpha$ that are $\alpha$-exposed from two different directions. Similarly, there exists a correspondence between the vertices of $S_\alpha$ and the circular arcs of $\partial U_\alpha$. Observe that the degree measure of an arc of $\partial U_\alpha$ is equal to the outward angle around the corresponding vertex of $S_\alpha$ (*Figure* 4.20).

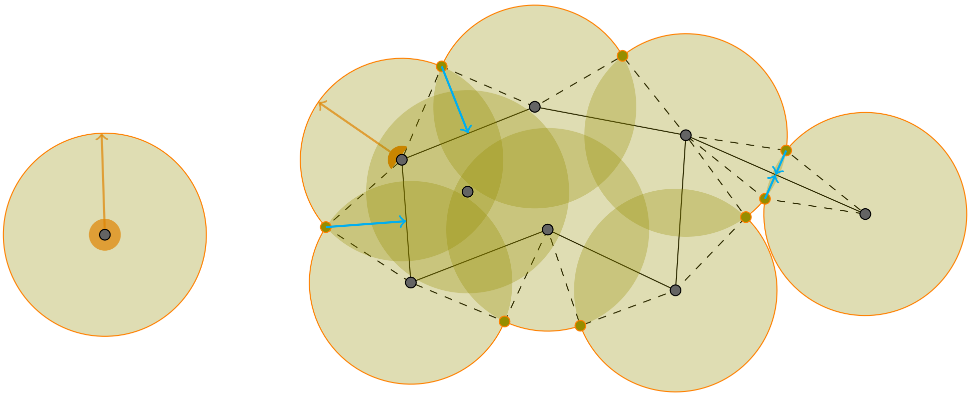


**Figure 4.20:** $\alpha$-diagram of the point set: the dashed lines indicate the mapping between the vertices and edges of the $\alpha$-shape and arcs and corners of the $\alpha$-diagram, respectively. Notice that this mapping is not a bijection without taking the multiplicities of edges that are $\alpha$-exposed from two different directions, such an edge is displayed on the right side of the figure which is mapped to two corner points.

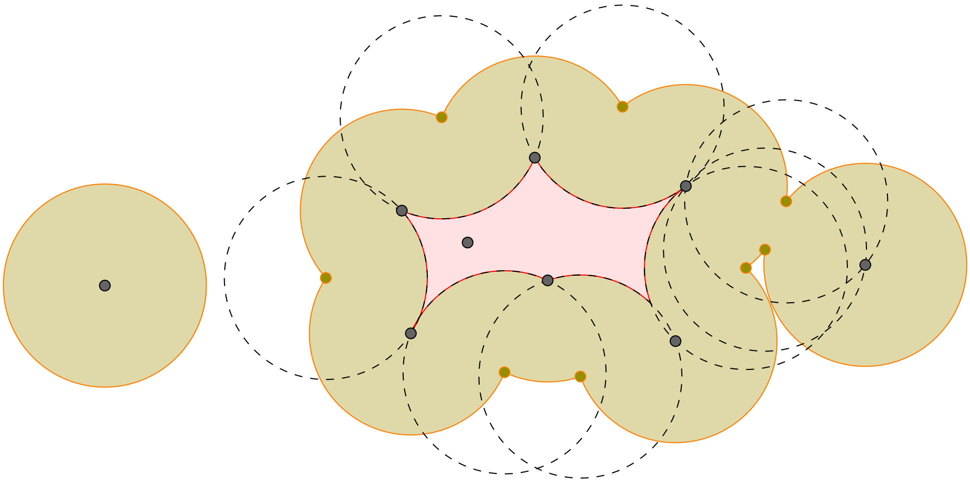


**Figure 4.21:** $\alpha$-hull of the point set is superimposed on its $\alpha$-diagram.

***Voronoi decomposition:*** Recall that the set of vertices and edges of an $\alpha$-shape is a subset of Delaunay triangulation. In fact, we will see that this statement can be generalized for $\alpha$-complexes. That is, an $\alpha$-shape is the underlying space of a simplicial complex which is a subcomplex of DT(S). Recall also that Delaunay triangulations are dual to Voronoi diagrams such that each Delaunay vertex, edge and triangle, corresponds to a Voronoi region, edge, and vertex respectively. Therefore, there exists a Voronoi region, edge or vertex for every simplex of an $\alpha$-complex.

Let $d_p$ be the $\alpha$-disk centred at point $p$ and $vor(p)$ be the Voronoi region of $p$. Observe in the *Figure* 4.22 that $vor(p) \cap \alpha$-diagram $= d_p \cap \alpha$-diagram. This observation leads to the following alternative construction of the $\alpha$-complexes and hence $\alpha$-shapes: the $\alpha$-complex of a point set for a fixed $\alpha$ can be constructed by taking the union of all Delaunay triangles, edges and vertices whose dual Voronoi cells have a non-empty intersection with the $\alpha$-diagram. Moreover, $alpha$-diagram is decomposed into convex regions by the Voronoi diagram (*Figure* 4.23).

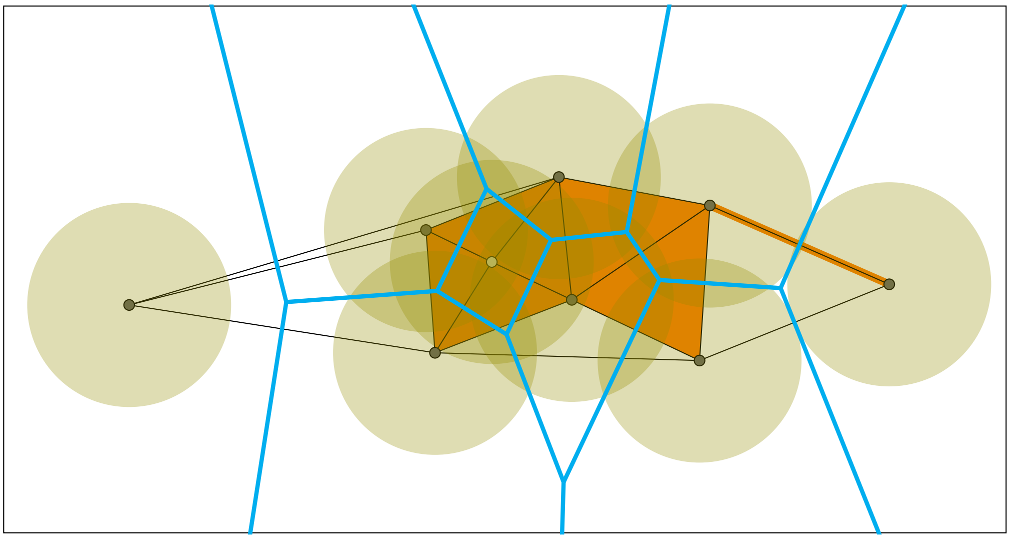


**Figure 4.22:** Voronoi diagram, Delaunay triangulation, $\alpha$-complex and $\alpha$-diagram: $\alpha$-complex can be constructed by taking the boundary of the union of all Delaunay triangles, edges and vertices whose dual Voronoi cells have a non-empty intersection with the $\alpha$-diagram.

***$\alpha$-complex:*** Assume that $\sigma_T$ is the k-simplex of a point set $T \subseteq S$ with $|T| = k+1$ and $k \in \{1, 2\}$. For each $\sigma_T$ we can define an open disk $d_T$ bounded by the smallest circle $\partial d_T$ that contains all points of $T$. Let $\rho_T$ be the radius of $d_T$. For $k = 2$, $\partial d_T$ is the circumcircle of $\sigma_T$ and for $k = 1$, the two points in $T$ are antipodal on $\partial d_T$. Let's define two sets $G_{k,\alpha}$ of k-simplices $\sigma \in DT(S)$ for which $d_T$ is empty and $\rho_T < \alpha\,(0 < \alpha < \infty)$. In addition, we define $G_{0,\alpha} = S$. Notice that, the union of $G_{k,\alpha}$ may not be a simplicial complex since not all edges of a triangle in $G_{2,\alpha}$ are necessarily in $G_{1,\alpha}$. The $\alpha$-complex of $S$, denoted by $C_\alpha$, is defined as a simplicial complex whose k-simplices are either in $G_{k,\alpha}$ or they bound (k+1)-simplices of $C_\alpha$. In fact, the second rule states that a k-simplex belongs to $C_\alpha$ if it is a face of a simplex in $C_\alpha$. By definition, for any $\alpha$, $C_\alpha$ is a subcomplex of $DT(S)$. In addition, $C_{\alpha 1}$ is a subcomplex of $C_{\alpha 2}$ if $\alpha_1 \leq \alpha_2$ and $\lim_{\alpha \to \infty} C_\alpha = DT(S)$ (*Figure* 4.24).

Based on $C_\alpha$, another definition of $S_\alpha$ can be given. The union of all simplices of $C_\alpha$

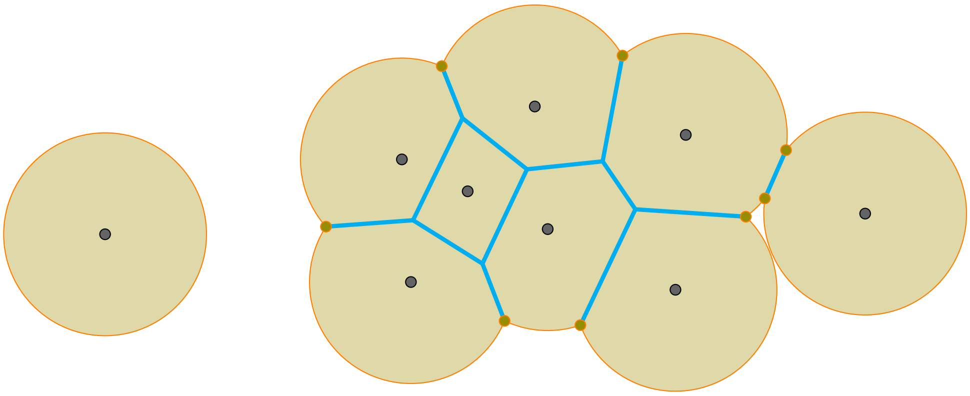


**Figure 4.23:** Voronoi decomposition of the $\alpha$-diagram.

defines a polytope which is equal to $S_\alpha$. This definition is more useful than the one that is based on the $\alpha$-exposed k-simplices since it leads to an easier implementation. In the former definition, it is not clear how to implement the inspection of a point whether it is $\alpha$-exposed or not; since there exist infinitely many $\alpha$-disks touching a 0-simplex. Conversely, the $\alpha$-complex definition provides an algorithmic approach.

Edelsbrunner and Mücke [1994] associated an interval for each simplex of $DT(S)$ indicating for which values of $\alpha$ the simplex belongs to $C_\alpha$. For instance, let $\sigma_T \in DT(S)$ be a k-simplex and $I_T = (t, \infty)$ be the associated interval. Then $\sigma_T \in C_\alpha$ if and only if $\alpha \in I_T$. Furthermore, Edelsbrunner and Mücke [1994] also partitions $I_T$ into three subintervals for which $\sigma_T$ is called an interior, singular or regular simplex. Please refer to the paper for computation of the intervals and for a detailed explanation of the algorithm. *Table* 4.1 summaries the outline of the algorithm.

| Step | Description |
|---|---|
| 1. | Compute $DT(S)$. |
| 2. | Compute an interval for each simplex in $DT(S)$. |
| 3. | For a given $\alpha$, compute $C_\alpha$. |
| 4. | All 2-simplices in $C_\alpha$ constitute the interior of $S_\alpha$. |
| 5. | All k-simplices with $k \in \{0, 1\}$, which are on the boundary of $C_\alpha$ form the boundary of $S_\alpha$ $(\partial S_\alpha)$ |

**Table 4.1:** Outline of the $\alpha$-shapes algorithm given in Edelsbrunner and Mücke [1994].

The advantage of this algorithm is that, step 1 and step 2 are independent from the value of $\alpha$ and it allows to compute the $\alpha$-shape of a point set with different $\alpha$ values efficiently. In fact, one of the main limitations of the $\alpha$-shape representation is related to finding the relatively best value of $\alpha$ for a given point set and a given application. In practice, this is mostly done by changing $\alpha$ iteratively and observing the output. Therefore, once the steps 1 and 2 are computed, only the rest of the algorithm is executed for varying $\alpha$ values. In addition, for some point sets, it might not be possible to find a *"satisfying"* value for the $\alpha$. Most of the time, such cases occur if the points are not uniformly sampled [Teichmann and Capps, 1998].

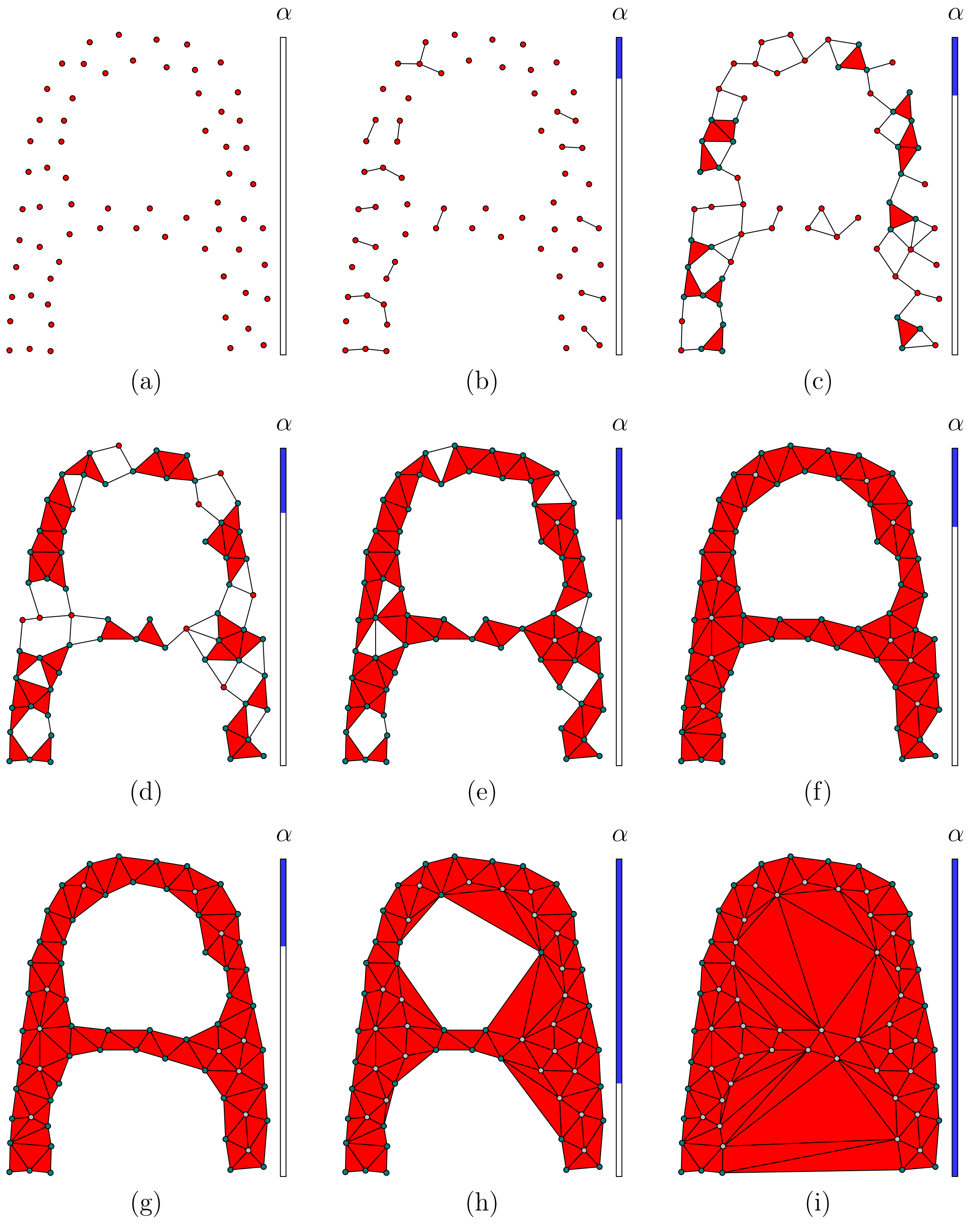


**Figure 4.24:** $\alpha$-complexes of a point set for different values of $\alpha$: (a) $\alpha \to 0$, (i) $\alpha \to \infty$, (b) - (h) in between values in increasing order.

$\alpha$-shapes have been used in many research fields such as computer vision and GIScience. Interested reader may refer to some articles in which shapes of objects are computed using $\alpha$-shapes: Park et al. [2005], Höfle et al. [2007], Carette et al. [2008] and Shen et al. [2011]. Moreover, there exist a section in Edelsbrunner and Mücke [1994] on the applications of $\alpha$-shapes.

The implementation of $\alpha$-shapes within the context of this work is discussed in *Section* 5.6.

## 4.3 Skeleton Operators

A skeleton operator decomposes a polygon $P$ into subregions with a tree structure inside $P$. There are two fundamental skeleton operators: medial axis (*Figure* 4.25) and straight skeleton (*Figure* 4.30). Medial axis is based on the Euclidean distance function such that it is composed of all interior points of $P$ whose closest point on the boundary of $P$ is not unique. Straight skeleton on the other hand, defined by an edge shrinking process in which the edges are moving inwards at a constant rate. Both operators have been used in numerous applications such as [Tagliasacchi, 2013; Gold and Dakowicz, 2005; Stefan, 2011; Siddiqi and M. Pizer, 2008].

### 4.3.1 Medial Axis

The idea of medial axis was first developed by Blum [1967] as a transformation of a general closed 2D shape into 1D curves. The medial axis of a 2D shape can be extracted by fitting maximally inscribed disks within the shape (*Figure* 4.26). The locus of all centres of these disks and their radii provides the Medial Axis Transformation (MAT) which is unique and invertible. That is, the original shape can be reconstructed from the centres and the radii of the maximally inscribed disks.

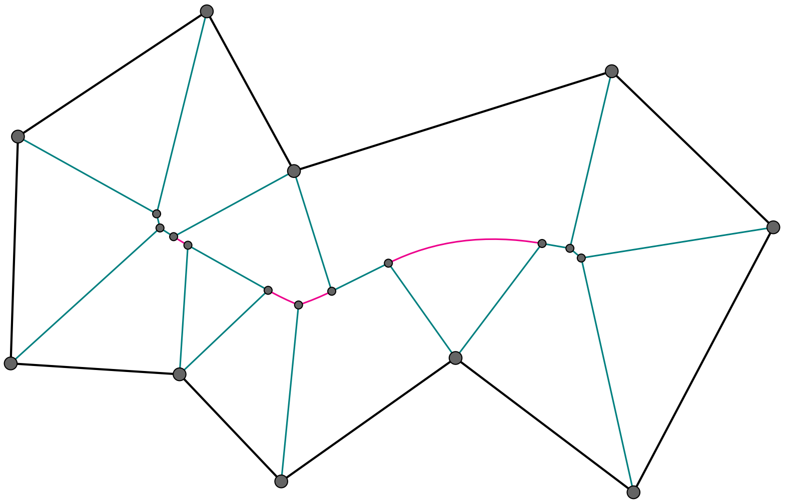


**Figure 4.25:** Medial axis of a polygon: linear segments and parabolic segments are displayed in different colors.

Medial axis can also be defined by an analogy to a grass fire. Imagine a uniformly grass-covered region bounded by a planar closed curve is set on fire at the same time everywhere along the boundary. As the fire propagates isotropically towards the interior, i.e. the fire grows at constant speed in every direction along the normals of the curve, some of the flames extinguish where the fire meets itself. These quench points will always be equidistant from the boundary. In other words, medial axis is composed of a point set in which the points have at least two closest points on the bounding curve. The analogy

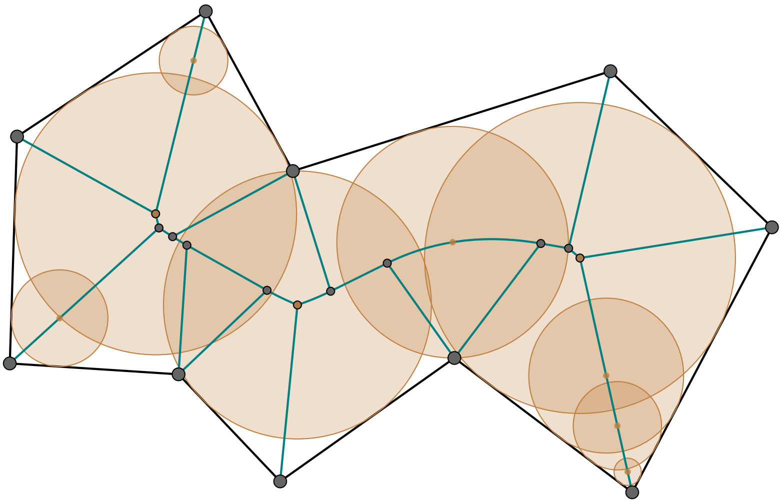


**Figure 4.26:** A subset of maximally inscribed disks in a polygon. The centres of all maximally inscribed disks constitute the medial axis of a shape. Observe that the degree of a medial axis vertex indicates how many times the maximally inscribed disk located on that vertex touches to the boundary of the polygon.

of grass fire reveals a close relationship between the medial axis and the Voronoi diagrams [Aurenhammer and Klein, 2000]. In fact, constructing a medial axis can be reformulated as a Voronoi diagram problem due to the fact that each Voronoi edge is the locus of points that have two closest neighbours within a given set of sites and each Voronoi vertex is the locus of points that has at least three closest neighbours within the sites.

Constructing medial axis of a shape bounded by a general plane curve may require solving systems of higher degree algebraic equations. Although it is possible to compute medial axis of these shapes exactly, in practice it is not convenient and approximate solutions are preferred. Furthermore, in many cases object shapes are not known exactly and they have to be approximated. Nevertheless, there exists some work on the exact computation of medial axis. For instance, Tzoumas [2011] presented a method for calculating the medial axis of a shape bounded by quadratic Non-Uniform Rational Basis Spline (NURBS) curves.

To approximate the medial axis of a general shape, the boundary of the shape has to be sampled either by points or linear segments. If only the points are used in the sampling, then the Voronoi diagram will be composed of lines and line segments (*Figure* 4.27). On the other hand, sampling with linear segments, i.e., approximating the boundary of the shape with a polygon, may produce parabolas, parabola segments, lines and line segments in the Voronoi diagram. *Figure* 4.25 displays the medial axis of a polygon extracted from the SVD of the polygon by filtering out the Voronoi vertices and edges outside the polygon. In a SVD, parabola segments are generated due to the reflex vertices[1] of a polygon. Note that, this is due to the fact that the locus of equidistant points from a fixed line (directrix) and a fixed point (focus) is a parabola (*Figure* 4.28) (Reflex vertices are likely to be the closest points to the edges on the opposite side of the polygon).

Many algorithms have been proposed for medial axis construction, most of which depend on the Voronoi diagrams. Please refer to Siddiqi and M. Pizer [2008] and Tagliasacchi [2013] for a broad survey. Here only two of them are mentioned. Chin et al. [1999]

[1] The internal angle of a reflex vertex is greater than $\pi$.

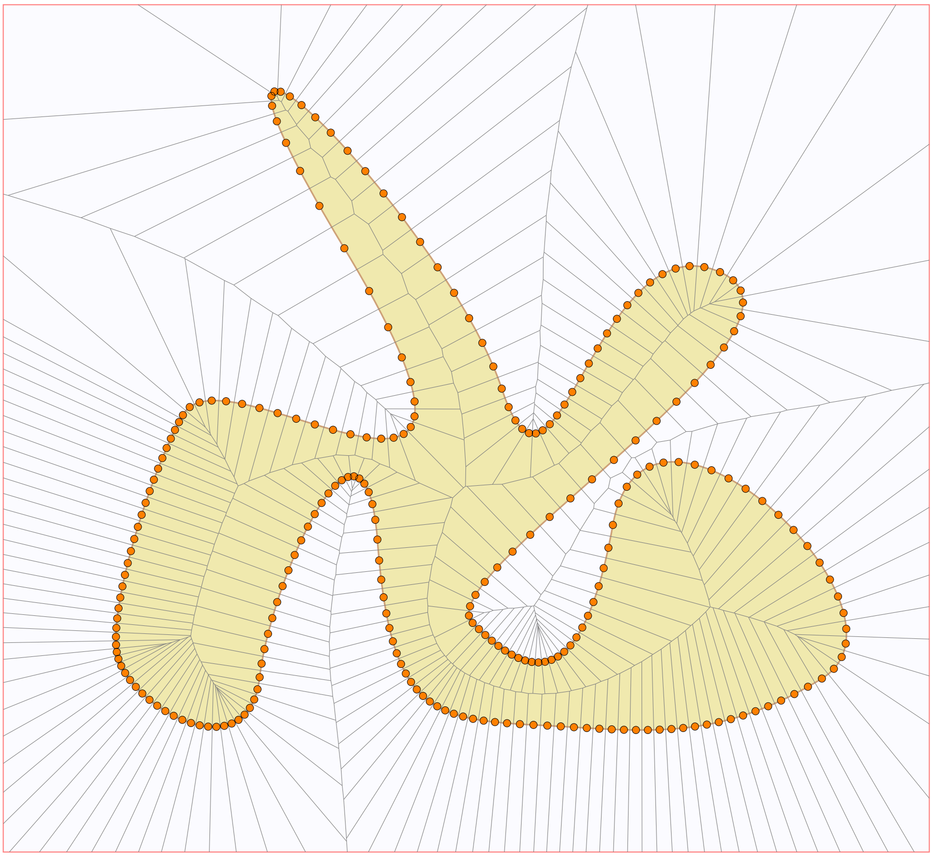


**Figure 4.27:** Approximating the medial axis of a smooth curve via Voronoi diagram.

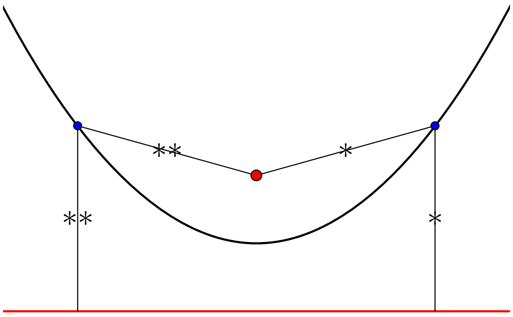


**Figure 4.28:** The locus of points equidistant to a given line (directrix) and a point (focus) constructs a parabola.

presented the first linear time algorithm for simple polygons[2] and Karavelas [2004] demonstrated an algorithm to construct SDGs which also allows intersecting sites. SVD is dual to SDG and medial axis can be extracted from SVD.

The concept of medial axis can be generalized to higher dimensions. Culver et al. [2004] presented methods for exactly computing the medial axis of polyhedrons in 3D and Musuvathy et al. [2011] introduced the medial axis of 3D shapes bounded by $C^{(4)}$-smooth parametric B-spline surfaces.

Medial axis has been used in many diverse application fields such as body animation in computer graphics, motion planning in robotics, domain decomposition in mesh genera-

[2] A simple polygon's edges only intersect at the vertices of the polygon.

tion, feature extraction in geometric design, tool-path creation in computer-aided manufacturing, medical shape analysis, shape correspondence and image analysis for shape recognition in computer vision [Siddiqi and M. Pizer, 2008; Tagliasacchi, 2013]. In addition, Gold and Dakowicz [2005] published a survey on the applications of medial axes in GIScience. They give the following application categories: text recognition and topology from scanned maps, object separation, skeleton retraction, terrain modelling and flow modelling and hydrography.

In *Chapter* 6 we proposed a method for generating exact pedestrian networks which depends on the creation of walkway centrelines using the medial axis. We first used the Karavelas [2004] algorithm for generating the SDG of a polygon with holes which models the surface of a walkway. Then, we extracted the centrelines from the SVD (dual to SDG) by simply filtering out the Voronoi edges that are outside the polygon with holes and the ones touching the boundary of the polygon with holes (*Figure* 4.29).

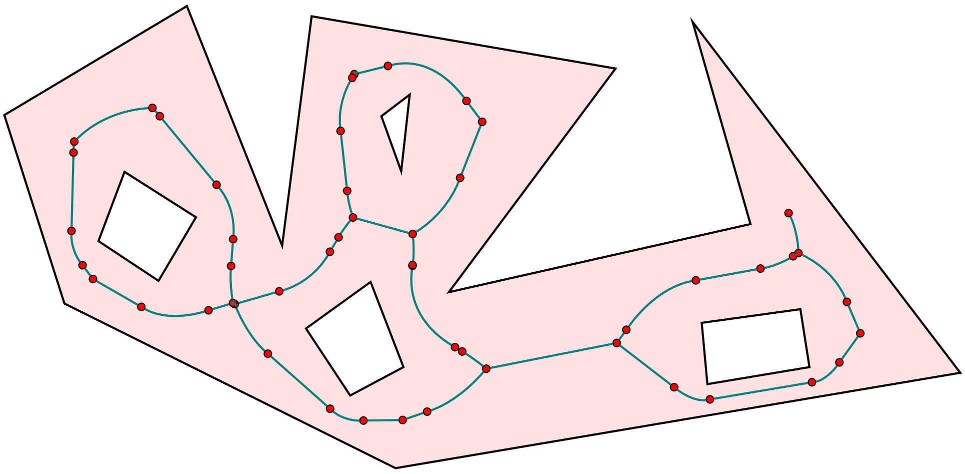


**Figure 4.29:** Centreline extraction with medial axis for an arbitrary polygon with holes.

### 4.3.2 Straight Skeleton

The notion of the straight skeleton of a polygon is introduced by Aichholzer et al. [1996] by an edge-parallel shrinking process. In this process, the edges are moving inwards at a constant rate while the vertices are kept on the angle bisector of the incident edges. This process continues until a topological change happens on the boundary of the polygon. There are two types of possible changes:

***Split event:*** When a reflex vertex (concave vertex) interferes with an edge, the edge splits into two which also results in the split of the whole polygon into two. New adjacencies occur between the split edge and each of the edges incident to the reflex vertex (*Figure* 4.31).

***Edge event:*** An edge collapses to length zero, making its neighbouring edges adjacent. If the lengths of the neighbouring edges also diminish at the same time, then all of the edges collapse into a single vertex. For instance, a triangle collapses into the center of the triangle's in-circle[3] (*Figures* 4.31 and 4.32).

[3] The in-circle center of a triangle is the intersection of the angle bisectors.

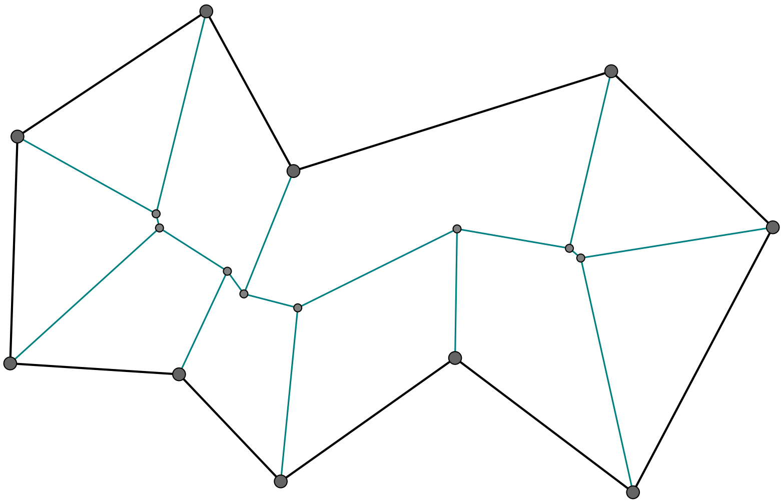


**Figure 4.30:** Straight skeleton of a polygon. Straight skeleton partitions the polygon into smaller polygonal regions such that for each polygon edge, there exists a polygonal region in the partition, called edge face.

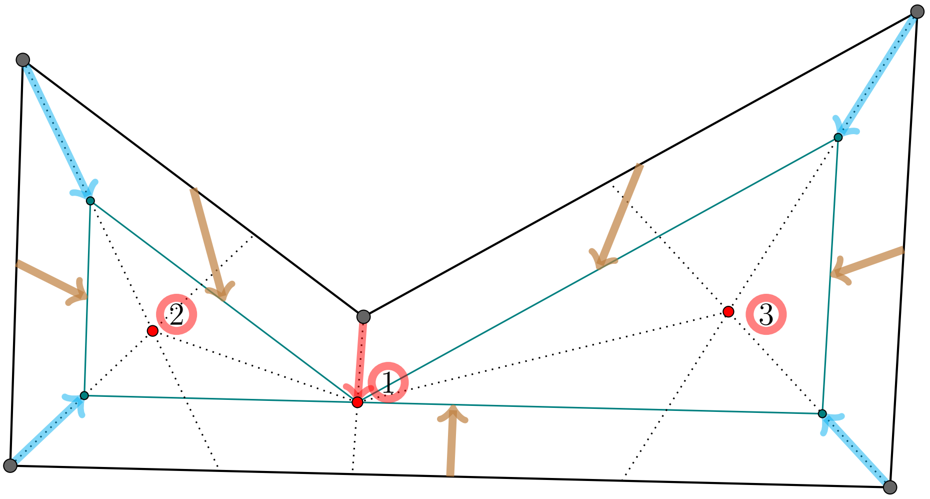


**Figure 4.31:** Events that happened during the straight skeleton edge shrinking process are marked in the order of happening. The first event is the split event in which a reflex vertex runs into a polygon edge and splits it. The second and third events are edge events in which the triangles are collapsed to a single vertex. Since the edges are moving at constant speed and the vertices are translating on the angle bisectors, the edges of a triangle collapse at the same time at the center of the triangle's in-circle.

An edge event either collapses the entire polygon into a point or it yields a new polygon which has one edge less than the previous one. Similarly, a split event results in two polygons. The edge-shrinking process is applied to all polygons in parallel until all of the polygons diminish and collapse into points. During this process, a hierarchy of nested polygons are generated (*Figure* 4.33). Furthermore, skeleton vertices or nodes are generated at every event position (where the events occur). Then, the straight skeleton is constructed from the union of the pieces of angular bisectors traced out from the polygon vertices to skeleton vertices. That is, the edges of a straight skeleton are constructed

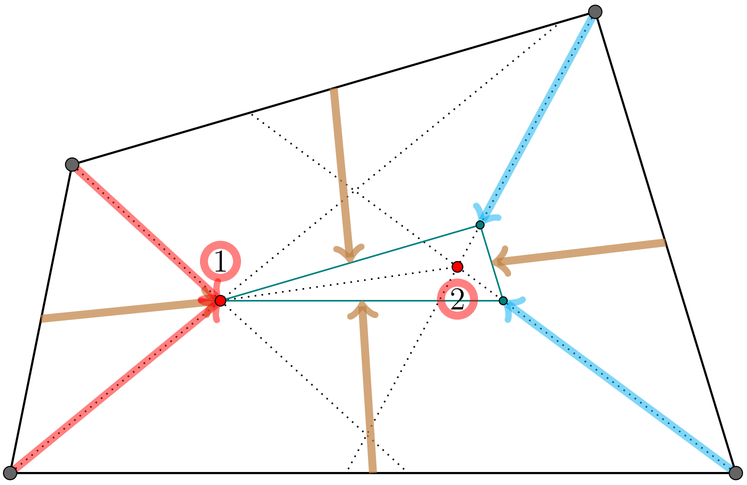


**Figure 4.32:** Events that happened during the straight skeleton edge shrinking process are marked in the order of happening. Both events are edge events. In the first event, the edge on the left diminishes and in the second event the triangle collapses into its in-circle center.

between the skeleton vertices which lie on the intersections of the angle bisectors[4] of the polygons. In addition, straight skeleton partitions the input polygon such that each polygon edge sweeps out a certain area called the face of the edge (*Figure* 4.30).

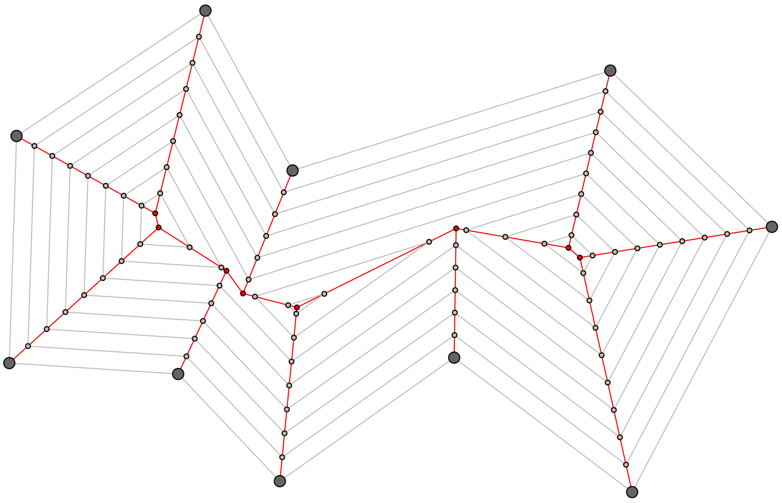
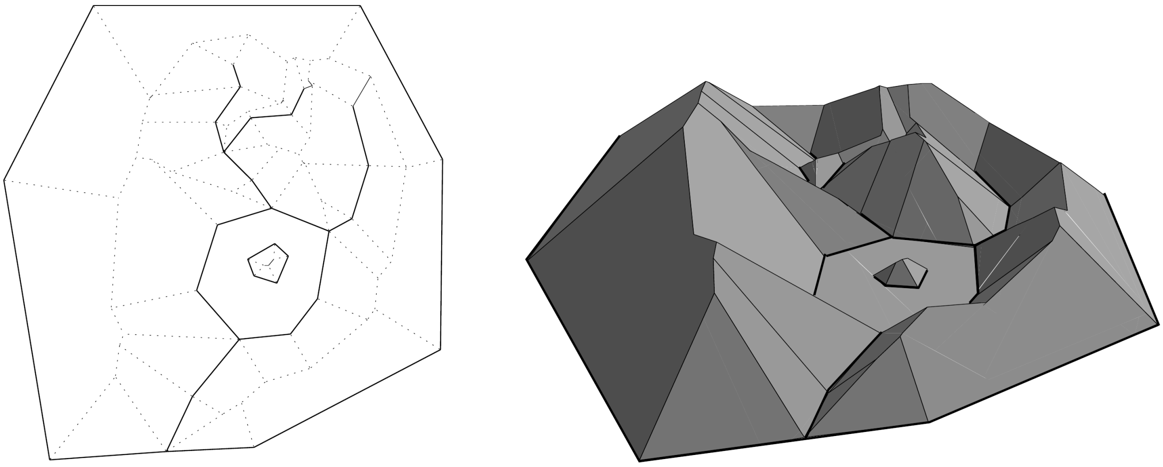


**Figure 4.33:** Straight skeleton and sample offset polygons generated during the edge shrinking process.

The definition of straight skeleton leads the implementation efforts. The first algorithm given by Aichholzer et al. [1996] is quadratic in run time complexity which simulates the shrinking process discretely. Later, Eppstein and Erickson [1999] developed a sub-quadratic algorithm by partitioning the problem into two sub-problems. The first and difficult problem is related to the interaction of reflex vertices. They solved this first sub-problem by generating graphs called motorcycle graphs and the second sub-problem

[4] The number of intersecting bisectors is 3 when a triangle collapses and it can be more than 3 when a regular n-polygon collapses ($n > 3$).

is related to the construction of straight skeletons from the motorcycle graphs. Recently, Cheng et al. [2014] improved this two step algorithm which was claimed to be the best algorithm in terms run time complexity. Interested readers should refer to Cheng and Vigneron [2007], Stefan [2011] and Cheng et al. [2014] for a detailed description of the general algorithm steps and a survey of the existing algorithms.



(a) coastline and river map

(b) reconstructed terrain from the map

**Figure 4.34:** Terrain reconstruction from a coastline and river map using straight skeleton. The image is taken from Aichholzer and Aurenhammer [1996].

Aichholzer and Aurenhammer [1996] extended the definition of straight skeletons and introduced the straight skeleton for general planar graphs in the plane. They showed that such an extension is very useful for terrain reconstruction from a coastline and river map (*Figure* 4.34). Furthermore, it is also well known that straight skeletons are very useful in computer graphics and GISs for generating plausible roof structures over the footprints of buildings [Eppstein and Erickson, 1999; Laycock and Day, 2003] (*Figure* 4.35). Finally, straight skeletons have found some other applications such as in mathematical origami they are used to find solutions to *fold and cut problem* [Demaine et al., 2000]; in solid modelling and Computer Aided Design (CAD), offset polygons are generated using straight skeletons [Sang and Yun, 2003] and in computer vision straight skeletons are used for 2D shape description and matching [Aichholzer et al., 2004].

Although medial axis algorithms are known to generate geometrically true centrelines for a road network, straight skeleton can also be used for this purpose [Haunert and Sester, 2008]. *Figure* 4.36 displays extracted centrelines for an arbitrary polygon with holes by filtering out the skeleton edges that are touching the boundary of the polygon.

### 4.3.3 Medial Axis and Straight Skeleton Comparison

Straight skeleton of a polygon is defined by the inward movements of the polygon edges at a constant speed along the normals of the edge segments. Similarly, medial axis can also be defined by the inward movements of points on the boundary of the polygon along point normals (remember the grass fire analogy mentioned in *Section* 4.3.1). Notice that the two definitions are quite similar. In fact, the straight skeleton and the medial axis of a convex polygon are completely equivalent since the edge bisectors are equidistant from

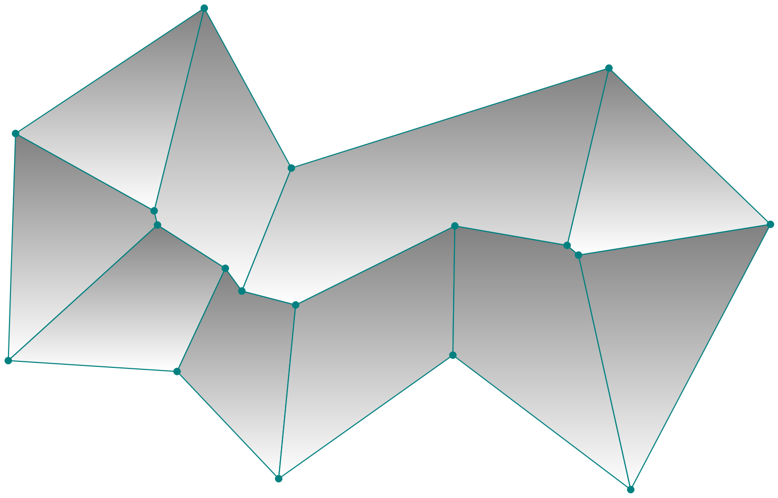


**Figure 4.35:** A roof structure constructed upon straight skeleton

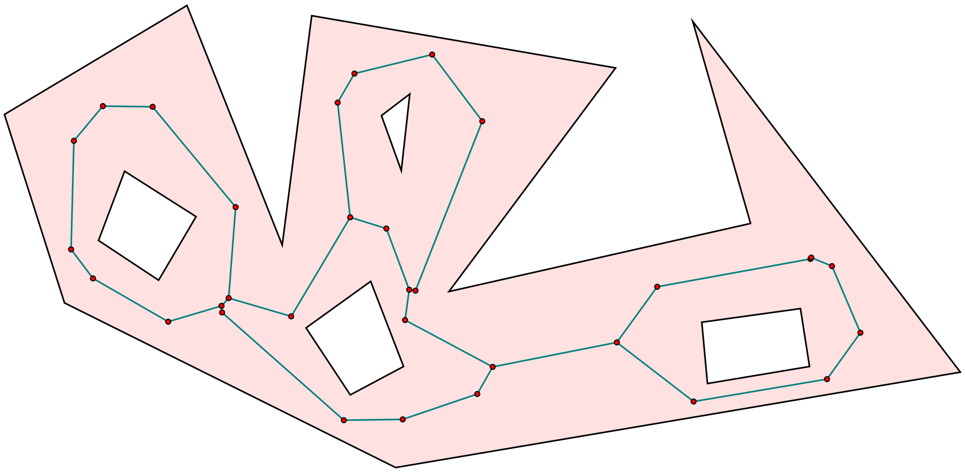


**Figure 4.36:** Centreline extraction with straight skeleton for an arbitrary polygon with holes

the edges and no split event occurs in the straight skeleton edge shrinking process for a convex polygon.

The two structures on the other hand differentiate for the concave polygons due to reflex vertices. Reflex vertices result in parabolic arcs in medial axis and split events in straight skeleton. A split event deviates the skeleton structure from a Voronoi diagram. However it avoids the parabolic arcs and keeps the structure linear (*Figures* 4.37 and 4.38).

Both medial axis and straight skeleton have the same limitation. They are scale insensitive such that the addition of an arbitrarily small geometric feature on the boundary of a polygon results in a large change on the structure. This stability problem may be significant in practical problems when there exist geometric variations on the boundary of the shapes due to noise. Please refer to Attali et al. [2009] for a review of the stability problem for medial axis and some possible solutions.

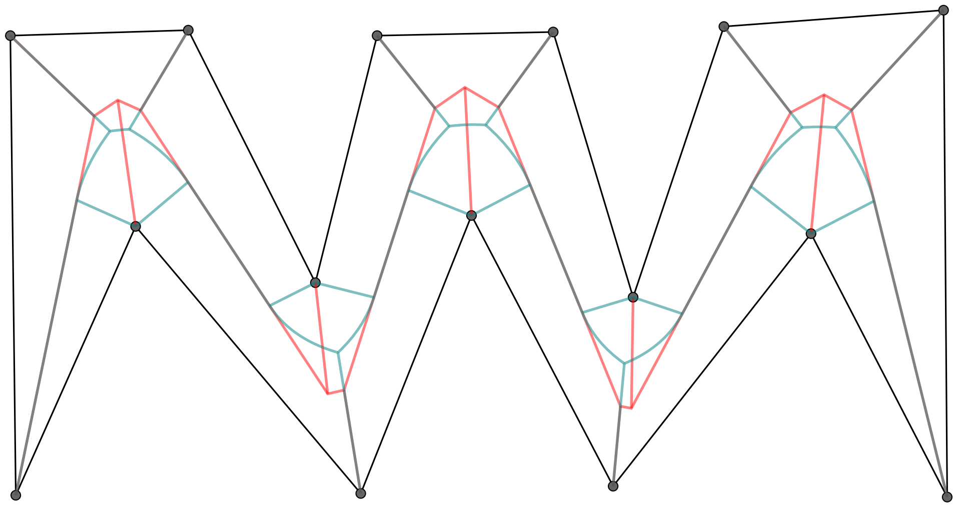


**Figure 4.37:** Medial axis (dark cyan) versus straight skeleton (red): reflex vertices cause the deviation between straight skeleton and medial axis.

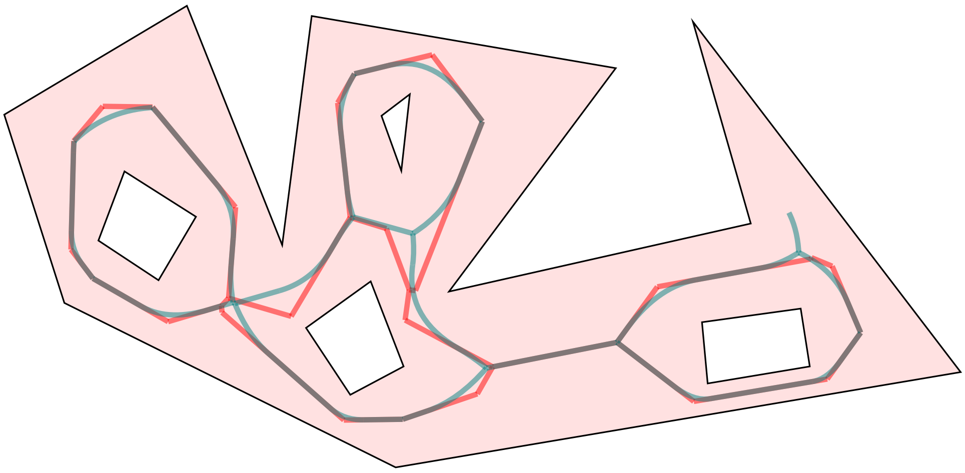


**Figure 4.38:** Centrelines generated with medial axis (dark cyan) and straight skeleton (red)

## 4.4 Connectivity Graph for Itinerary Calculations

We developed the idea of connectivity graphs to benefit as much as possible from the created semantic planar partitions (semantic objects + planar partition). The idea is to have additional thematic graphs that can be used besides the navigational network graphs to assist for the itinerary calculations.

We compute the connectivity graphs form the semantic planar partitions. A graph node is created for each traversable / crossable borderline between two semantic objects. The crossable property of a borderline is application specific that might depend on several conditions such as the type of the travelling entity, the types of semantic objects on both sides of the borderlines and some application specific rules. For instance, a borderline between a sidewalk and a pedestrian crossing is traversable by a pedestrian (type of the entity) if the green light is on for the pedestrian network (application specific rule). In this example, the application specific rule is a dynamic one which disables the corresponding node from the connectivity graph when the application rule is not satisfied. The edges of the connectivity graph encodes the connectivity information between semantic objects[5]. That is, an edge between two borderlines indicates a navigable region on the surface map which can be used by a travelling entity. Thus, a connectivity graph as a whole represents a subset of connected semantic objects which form a continuous navigable surface. Connectivity graphs can either be directed or undirected. If borderlines are crossable in both ways, then the graphs are undirected, otherwise directed edges are needed which indicate the direction of allowed crossings.

More than one connectivity graphs can be generated on the same semantic planar partition. For instance, creating a separate connectivity graph for each type of entity might ease the application specific rules (there will be no rules depending on the type of the travelling entity). *Figure* 4.39 displays two connectivity graphs on the same semantic planar partition, one for walkways (e.g. for pedestrians) and one for roads (e.g for motorized vehicles).

## 4.5 Conclusion

In this chapter, a set of generic algorithms from the literature that can be utilized in many different GIS applications are discussed with theoretical and practical perspectives. In addition, we introduced the connectivity graphs computed on top of the semantic planar partitions which can be used beside the navigational network graphs for rule based itinerary calculations.

The discussed algorithms are integrated into our GIS framework called StreetMaker (*Chapter* 5). Having these algorithms ready at hand can significantly reduce the application development time. In fact, in *Chapter* 6 we utilized some of these algorithms for constructing a static obstacle avoiding pedestrian network: medial axis transform is used to compute the centrelines of the walkways and the shapes of the obstacles on the walkways are estimated using $\alpha$-shapes (which also depends on the Delaunay triangulations).

As the number of StreetMaker applications increases, there will be more and more generic algorithms integrated. For instance, generic graph search algorithms can be (e.g. A*, shortest path) integrated for itinerary calculations. Furthermore, 3D algorithms such as 3D Delaunay triangulation, 3D alpha shapes, 3D mesh/surface generation, etc. can also be integrated for developing 3D GIS applications.

[5] The edges do not correspond to actual geometric routes between the semantic objects. They just describe the topology between the nodes.

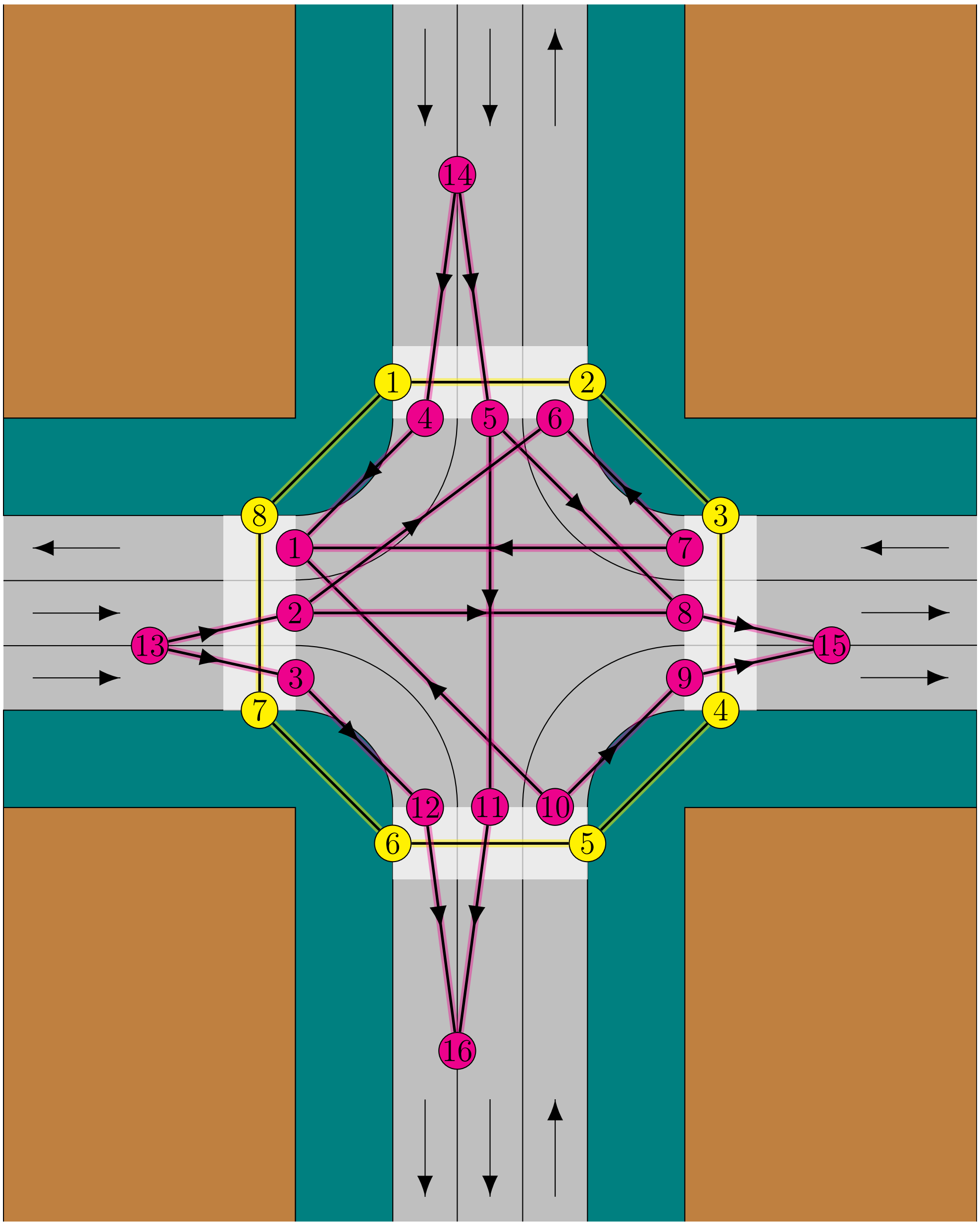


**Figure 4.39:** Connectivity graphs on a semantic planar partition: two connectivity graphs are displayed on a semantic partition composed of buildings (brown), sidewalks (teal), pedestrian crossings (white) and lanes (gray). The first graph (yellow) is related to the walkways, indicating the connections between sidewalks and pedestrian crossings. The second graph (magenta) is related to road networks, indicating the connections of the lanes. Observe that the second graph is directed, implying that object borders can only be crossable along the direction of the graph edges.

# Chapter 5

# StreetMaker - A Generic GIS Framework

## 5.1 Introduction

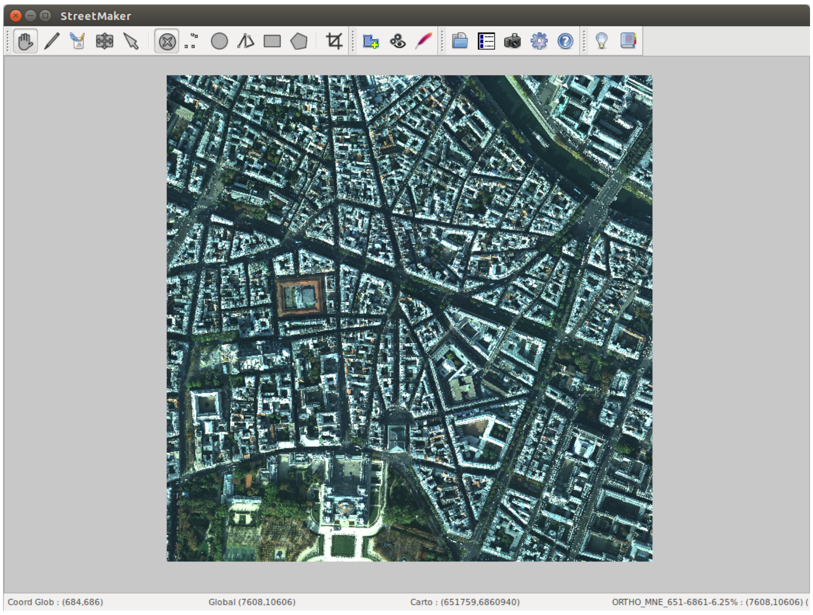
StreetMaker is a GIS framework designed for easy GIS application development. It is composed of three main components: (i) Graphical User Interface (GUI) and basic GIS capabilities (*Section* 5.2), (ii) the data model (*Section* 5.3) and (iii) generic algorithms (*Section* 5.6). Applications can be built upon StreetMaker which can utilize all of the three main StreetMaker components.

The key component of StreetMaker is its data model which is well-suited for modelling the geospatial objects using planar partitions with an object oriented approach. In this data model, planar partitions are encoded into topological 2D arrangements. Thanks to the underlying topological data structure (half-edge data structure) efficient and effective geometric computing is possible. This is achieved by explicitly storing the connectivity (topological) information between the geometric primitives of the planar partitions. Additionally, the exact computation paradigm (*Section* 5.5) is used for the implementation of the 2D arrangements for robust geometric computing.

Semantic objects representing the real life geospatial objects in computers are constructed either by grouping the planar partition elements (geometric primitives) or previously created objects. These objects capture semantic, thematic and geometric information related to the geospatial objects and the hierarchy among them are managed via DAG.

The GUI (*Figure* 5.1) and the basic GIS capabilities of StreetMaker is used for creating and editing planar partitions and constructing semantic objects on them. The generic functions integrated into StreetMaker can be utilized by any application built upon StreetMaker. Applications built on StreetMaker either create their own semantic planar partitions (planar partition and semantic objects) or utilize the existing semantic planar partitions that were created by other applications. If a semantic planar partition models the geospatial objects with fine details (fine object resolution at the lowest level and fine hierarchy), the possibility of re-usability increases.

## 5.2 StreetMaker GUI and Basic GIS Capabilities



Coord Glob : (684,686)    Global (7608,10606)    Carto : (651759,6860940)    ORTHO_MNE_651-6861-6.25% : (7608,10606) (

**Figure 5.1:** StreetMaker GUI displaying an orthophoto in the main window: aerial images that are overlaid at the background provide visual guides to human operators in processing the vector data.

StreetMaker enables human operators[1] to navigate within the multiple layers of raster and vector data. In a StreetMaker session, layers can be created, loaded, processed and saved back to disk. Below a list of possible operations that can be performed are given.

- Browsing the geometry and object hierarchy: Basic navigation tools (panning, zooming, etc.) allow operators to navigate within the data. Furthermore, not all objects might be visible in the main window due to the occlusion of the higher level objects. A separate list of objects and their hierarchy is supplied to the user in order to visualize the occluded objects (*Figure* 5.4). The same list can also be used for selecting objects and analysing and modifying the object hierarchy. Another list available to operators is the list of layers, on which the layers can be activated / deactivated and their order of rendering can be altered.

- Snapping: When inserting, deleting and selecting vector data, the mouse pointer is snapped to geometry (points and lines) at various zoom levels, so that typical GIS errors such as over-shoot and under-shoot are avoided.

- Selecting: Geometry (points, lines, polygons) and objects can be selected in various ways, by clicking on the geometric features / objects, by unique ids that are attached to each geometric feature / object and according to properties of the geometric features / objects, e.g., automatic selection of all holes inside selected polygons, selecting all isolated points, selecting objects according to their types, etc. The selected geometry and objects are ready to be processed with other means of operations such as deletion, copying, creating objects from them, etc.

[1] From now on, the words "operator" and "human operator" will be used interchangeably which in fact mean any user of the StreetMaker.

- Simplification and error correction tools: Some automatic and semi automatic tools are added to StreetMaker in order to ease the work of operators for fixing the geometric and topological errors within the vector data (*Section* 5.4.4).
- Merging, converting, importing and exporting vector layers: There exist two fundamental types of vector layer representing planar and semantic planar partitions. These layers are called Planar Partition Vector Layer (PPVL) (*Figure* 5.2) and Semantic Planar Partition Vector Layer (SPPVL) (*Figure* 5.3). External vector data can be loaded and imported into PPVLs and SPPVLs (*Sections* 5.4.2 and 5.4.3 respectively). Furthermore, two or more PPVLs can be merged into a single PPVL layer. That is, all the intersection points between the corresponding 2D arrangements are computed and merged into a new 2D arrangement. Finally, the geometry and the semantic information stored in SPPVLs via semantic objects can be exported to ESRI shapefile and XML file formats.
- Offsetting layers: Any vector layer can be offset and moved in the coordinate system. This feature is helpful when dealing with very big coordinate numbers. Especially when the calculations are done in exact computation (*Section* 5.5), small coordinate numbers avoid possible errors and increase the computing performance. Coordinates can be restored to original values by applying the same offset in the reverse order after the geometric processing is finished.
- Algorithms: Algorithms are executed on the activated layers or only on the selected geometry / semantic objects within the activated layers depending on the operator's choice (*Section* 5.6).

## 5.3 Data Model

In StreetMaker, data models are encapsulated in layers. There exists one raster layer and two vector layers. The raster layer is used to display images (generally aerial images) which give visual guidelines to the operators when manually processing the vector data (*Figure* 5.1). The data model described in *Chapters* 2 and 3 are employed as vector layers. PPVL encapsulates the 2D arrangement data structure, therefore representing the planar partitions[2] without any semantic data (*Figure* 5.2). SPPVL is derived from PPVL and it is the semantized version of PPVL. The semantization is done by creating semantic objects on top of the underlying arrangement cells. SPPVLs are capable of managing these semantic objects with their hierarchy (using DAGs) and performing object related functionalities such as creating, deleting, modifying etc., (*Figure* 5.3).

## 5.4 Creating Semantic Planar Partitions

In StreetMaker, PPVLs can be created by two different ways and so SPPVLs since any PPVL can be converted to a SPPVL by semantization. Additionally, depending on the data at hand, SPPVLs can also be created directly. That is, the geometry and the

[2] In fact, 2D arrangements represent planar graphs from which planar partitions can be extracted easily as described in *Chapter* 2.

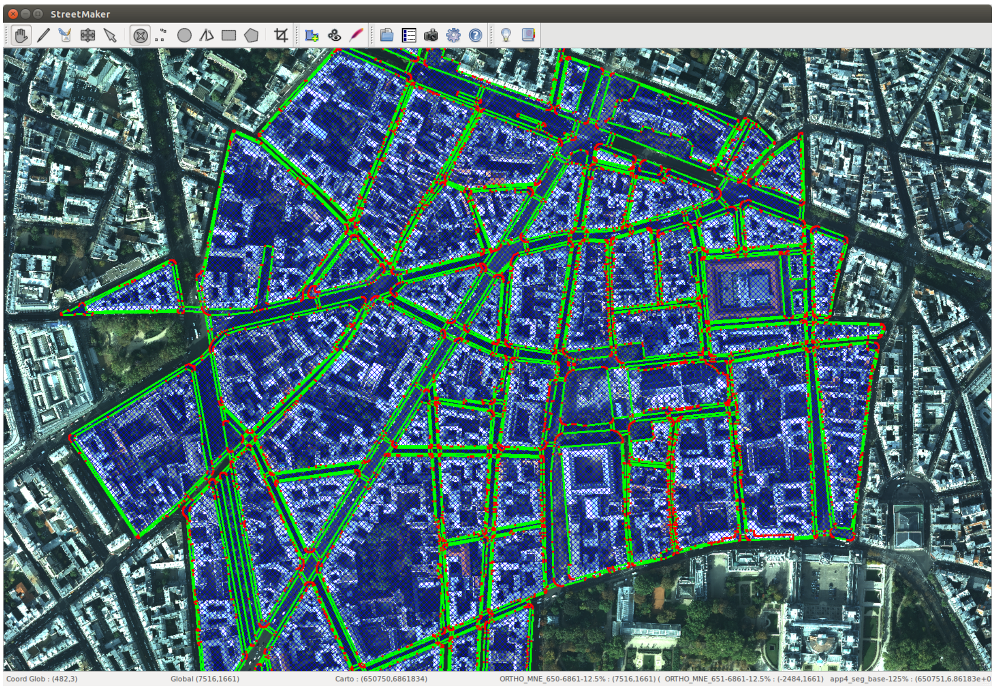

StreetMaker
Coord Glob : (482,3)
Global (7516,1661)
Carto : (650750,6861834)
ORTHO_MNE_650-6861-12.5% : (7516,1661) ( ORTHO_MNE_651-6861-12.5% : (-2484,1661) app4_seg_base-125% : (650751,6.86183e+0

**Figure 5.2:** A PPVL in StreetMaker: the underlying data structure for planar partitions is 2D arrangements.

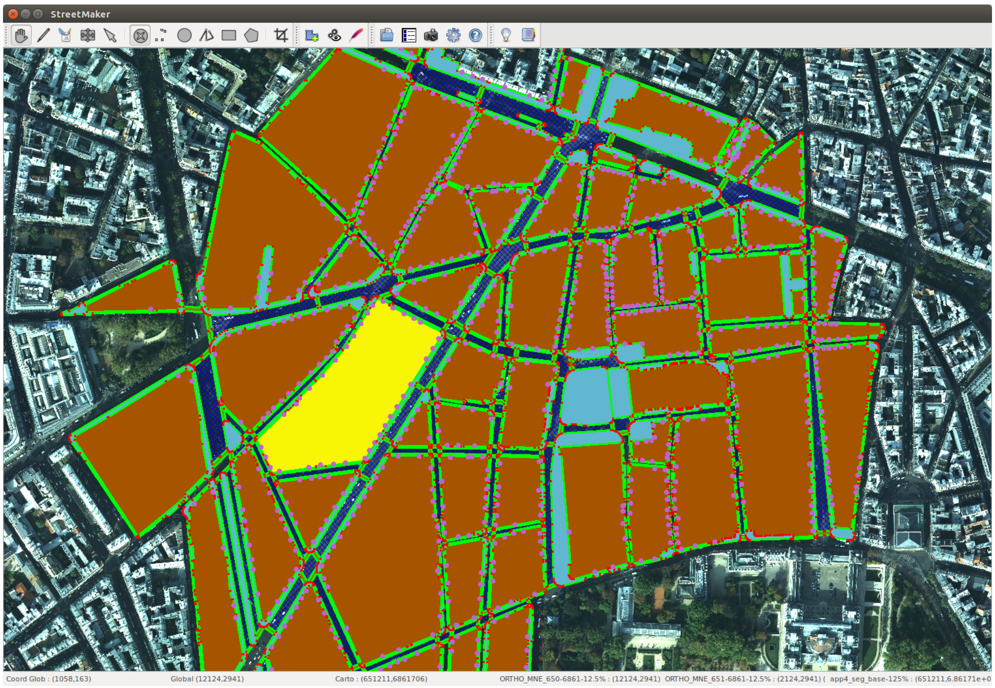
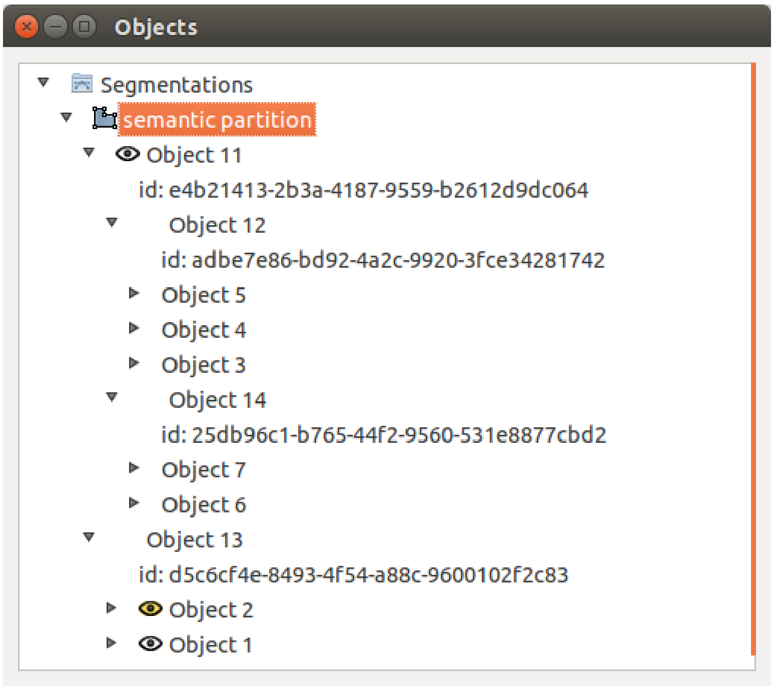

StreetMaker
Coord Glob : (1058,163)
Global (12124,2941)
Carto : (651211,6861706)
ORTHO_MNE_650-6861-12.5% : (12124,2941) ORTHO_MNE_651-6861-12.5% : (2124,2941) ( app4_seg_base-125% : (651211,6.86171e+0

**Figure 5.3:** A SPPVL in StreetMaker: SPPVLs are derived from PPVLs by semantization, i.e. by creating semantic objects on the arrangement cells. Different types of semantic objects are rendered in different settings which can be altered. In addition, a fixed set of settings has to be assigned for special cases. In the figure for instance, a selected object is highlighted in yellow according to the current fixed settings for the object selection.

semantic information related to semantic objects can be captured at the same time from an external data source into a SPPVL. The following three subsections discuss the three different methods mentioned here.



Objects

```
▼ Segmentations
  ▼ semantic partition
    ▼ Object 11
        id: e4b21413-2b3a-4187-9559-b2612d9dc064
      ▼   Object 12
          id: adbe7e86-bd92-4a2c-9920-3fce34281742
        ▶ Object 5
        ▶ Object 4
        ▶ Object 3
      ▼   Object 14
          id: 25db96c1-b765-44f2-9560-531e8877cbd2
        ▶ Object 7
        ▶ Object 6
    ▼   Object 13
        id: d5c6cf4e-8493-4f54-a88c-9600102f2c83
      ▶ Object 2
      ▶ Object 1
```

**Figure 5.4:** An object hierarchy list in StreetMaker: visible objects are displayed with an eye icon. The yellow eye icon (Object 2 in the figure) indicates that the corresponding object is selected. In the figure, Object 11 is parent to objects 12 and 14 which are also parent to objects 3, 4, 5 and 6, 7 respectively. Notice that parent and child objects cannot be visible at the same time. The visibility of objects can be modified (by a right click) on the object hierarchy list.

### 5.4.1 Semantic Planar Partitions from Scratch

If there is no data for the area of interest other than aerial photographs, PPVLs have to be created from scratch. Here is the procedure:

- The operator creates an empty PPVL and loads at least one aerial image (possibly an orthophoto) as a raster layer. The coordinate system used in aerial images are automatically used for the PPVL. Therefore, if the images are geo-referenced, then the geometry drawn on top of them will be also geo-referenced.

- The operator draws the outlines/footprints of the interested geospatial objects one by one on the raster layer. The geometry described by the user drawn polylines are continuously inserted into the corresponding PPVL and the underlying partitioned plane is automatically calculated by the 2D arrangements data structure associated to each PPVL.

- After the drawing step is finished (i.e. creation of PPVL is completed), the operator can start to create objects on the planar partition by selecting the arrangement cells. Although it is possible to create objects with any combination of arrangement cells, in general a couple of faces, edges and vertices are selected for areal objects, linear objects and punctual objects respectively. Furthermore, already created objects can be used to form the parent objects which builds up the object hierarchy. As a result, the PPVL is converted into a SPPVL.

### 5.4.2 Semantic Partitions from Unstructured Vector Data

Previously created existing vector data can be leveraged to create planar partitions. However, very few data can be used directly without pre-processing[3]. In many cases, the data in hand need to be fixed or completed due to topological and geometric errors as well as missing data. The procedure is described below.

- Vector data coming from different sources are imported into the system as PPVL(s). Two approaches are possible.

  1. All data are imported into a single PPVL.
  2. Different data coming from different sources are imported into separate PPVLs.

  The first approach is generally easier and preferable to the second one since all data are managed within a single vector layer. On the other hand, the second approach yields more structured initial sets of input vector data. Conceptually related data can be grouped into the same PPVLs and then the resulting PPVLs can be merged into a single one. Besides, if there are too many data sources, manually fixing the errors and completing the missing data might be difficult (too many unstructured data may be visually cumbersome) within a single PPVL.

- Geometric and topological errors existing in the PPVLs are fixed by the operator using the automatic and semi-automatic tools (*Section* 5.4.4) beside the manual corrections and completions (possibly utilizing an areal image overlaid at the background) of the missing data.

- Semantic objects are created. This step depends on the selected approach in the first step.

  1. The PPVL is converted to a SPPVL by creating semantic objects on the selected arrangement cells (see the final step of the procedure explained in *Section* 5.4.1).
  2. First, an empty SPPVL is created. Then, for each semantic object creation, the selected geometry from different PPVLs are copied into the initially created SPPVL. The intersections are automatically detected and handled at the geometry and object levels. The parent objects can be created within the SPPVL from the other semantic objects as discussed before.

### 5.4.3 Semantic Planar Partitions from Structured Vector Data

The simplest method for generating semantic planar partitions is using structured vector datasets when they are available. A structured vector dataset does not need to be modified and it already describes the outlines of geospatial objects without or with minimum amount of errors. Therefore, objects are automatically created (imported) without requiring any human interaction. The mapping between the geometric features and the automatically created semantic objects is configured according to any semantic information that exists in the input vector data. If there is no semantics, then all created objects will be of the same user defined type.

[3] A vector data that needs to be processed for fixing errors are called unstructured.

In this method, the quality of the obtained partitions depend on the quality of the input dataset. For instance, if two polygons representing two neighbour objects have tiny overlapping regions, these intersections result in tiny polygons that are attached to both objects in the corresponding SPPVL. If such cases are not acceptable, then the method described in *Section* 5.4.2 can be followed.

### 5.4.4 Handling Basic Geometric and Topological Errors

StreetMaker has a few standard automatic error fixing features[4]. Some of these features depend on a user defined threshold. Thresholds are not generic, they have to be adjusted according to data at hand.

#### 5.4.4.1 Redundant Geometry Removal

If the minimum angle ($\alpha$) between two consecutive edges is smaller than a predefined threshold ($\epsilon$), then the edges are merged by removing the in between shared vertex (*Figure* 5.5). Note that, the degree of the shared vertex has to be two (2).

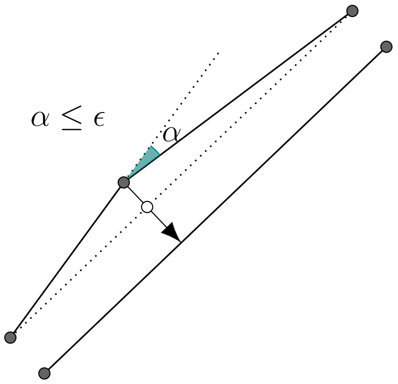


**Figure 5.5:** Redundant geometry removal based on an angle threshold ($\alpha \leq \epsilon$): the middle vertex is removed which results in a straight line segment.

#### 5.4.4.2 Auto Connecting Lines

Very close lines are common in many vector datasets because of under-shoot errors or coordinate mismatches. Depending on a distance threshold one of the two lines is extended to meet the other one (*Figure* 5.6).

[4] Similar auto-correcting tools exist almost in every GIS. Interested reader may refer to GRASS GIS (https://grass.osgeo.org/documentation/) and PostGIS Topology (http://postgis.net/docs/manual-dev/Topology.html) for examples of similar other tools.

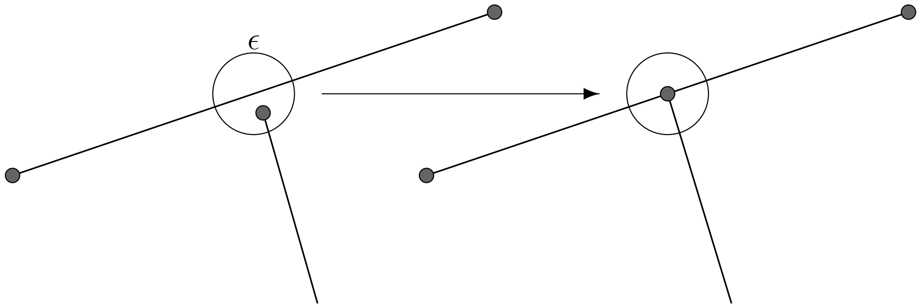


**Figure 5.6:** Auto connecting lines based on a distance threshold.

#### 5.4.4.3 Merging Vertices within a Neighbourhood

Vertices within the same neighbourhood defined by a relatively small disk are merged into their mean vertex. Notice that, in *Figure* 5.7, the merged vertices were not connected by an edge. In fact, this property can be configured by the operator to prevent merging of unconnected vertices.

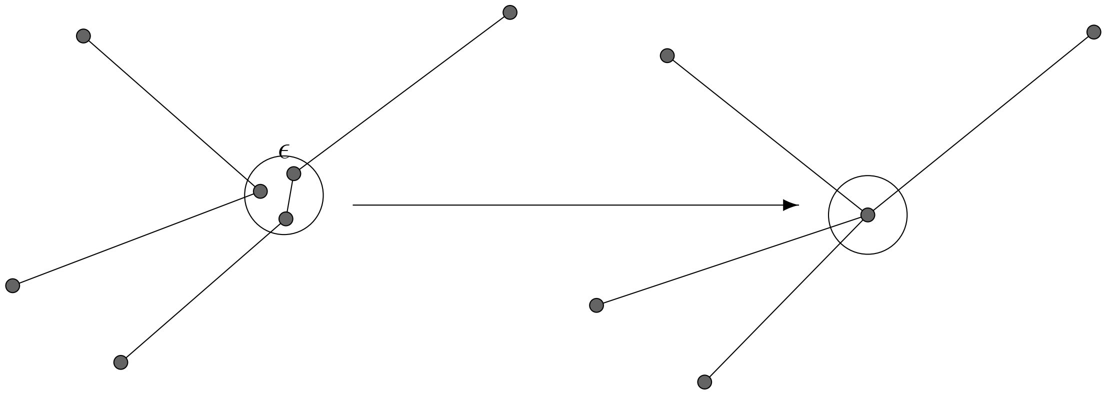


**Figure 5.7:** Merging vertices within a neighbourhood based on a distance threshold.

#### 5.4.4.4 Antenna Removal

Recall that for a valid planar partition, valid polygons are required (*Section* 2.3.2, *Table* 2.3). Therefore, before creating polygonal (areal) semantic objects, one needs to make sure that the used polygons are valid. The constraints of valid polygons have already been discussed in *Section* 2.3.2 (*Table* 2.4 and *Figure* 2.13). *Figure* 5.8 displays an example for converting an invalid polygon into a valid one by removing the antenna like structures, isolated vertices and dangling edges. In this figure, the triangular polygon on the top right corner touches the exterior ring of the polygon which is allowed by definition if the touching polygon is an inner boundary (i.e. hole). Therefore, depending on the interpretation (or design decision) internal polygons touching the exterior boundaries can be kept or deleted. In StreetMaker, these touching inner polygons are excluded from the exterior rings when creating semantic objects. In fact, if one wants to interpret these touching polygons as inner holes (i.e. as exterior regions to the polygons), creating a separate object (might be a dummy object which can deleted afterwards) from the touching polygon will do the job.

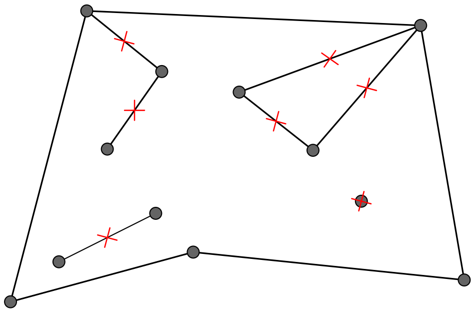


**Figure 5.8:** Removing antenna like structures: cleaning internal structures from the polygonal features to extract valid polygons.

## 5.5 Robustness in StreetMaker: Exact Computation Paradigm

Mathematicians classify the numbers as integers, rational numbers, irrational numbers, real numbers, complex numbers and so on. In computers these numbers are stored in binary format. Integers and rational numbers can be represented exactly without loss of data if there is no restriction on the memory. On the other hand, the definition of irrational numbers are problematic even in modern mathematics and hence in computing theory. These numbers cannot be expressible exactly and the best we can do is to approximate them with an infinite series of numbers. In many fields of engineering, approximate solutions are acceptable. Therefore, approximating real numbers with fixed length of bits is not a problem. However, due to the plenty of different representations it was likely that the same software could produce different output in different hardware. The community went for a standardization to overcome this problem with ANSI/IEEE Standard 754 [1985] [Overton, 2001]. On the other hand, in scientific computing and computational geometry for many applications the standard approximation to real numbers is not enough. Fixed-size floating points cause these programs to crash or at best result in errors in the outputs [Kettner et al., 2008].

Geometric constructions[5] and predicates[6] may require exactness. For instance, a point and a line can be in two different configurations with respect to each other: either the point is on the line or not. Finding an exact answer to this simple predicate may be very tricky if the point lies on the line or is very close to it. If the resolution of the point coordinates are not enough (e.g. when IEEE 754 floating point format is used), the answer that is calculated for this predicate may not be correct. Furthermore, not all decimal numbers are expressible exactly in binary format. For example, a very simple decimal number 0.1 cannot be represented exactly but is approximated: $(0.1)_{10} = (0.1001100110011001...)_2$. As a result of these observations, the fixed floating point representations are not sufficient to implement geometric algorithms which are generally designed with the assumption of exact geometric constructions.

A solution to mentioned robustness problems is using the variable size integers in rational number arithmetic and avoiding floating point numbers at all. However, this approach does not work for non-linear geometric problems in which the irrational numbers take

[5] Points, lines, circles, etc.

[6] Geometric queries related to configurations of geometric constructions with respect to each other, e.g. is a line tangent to a circle, does a point lie on/in/outside of a circle, etc.

part. For dealing with irrational numbers, the length of the fixed floating points are generally extended to create multiple precision floats to replace the standard single or double precision floats [Fousse et al., 2005]. This approach increases the accuracy of the approximations to arbitrary precision (generally user defined and memory bounded), however the difficulty is the determination of the precision that is needed. Moreover, one cannot be sure whether two numbers are exactly equal. The equality of two numbers can only be determined upto the limit of the user defined precision.

Numbers can also be classified into two groups: algebraic and transcendental numbers. Algebraic numbers can be expressible by the roots of a polynomial with rational coefficients. It is not difficult to see that all integers and rational numbers are algebraic. Besides, some of the irrational numbers are also algebraic. For instance, $\sqrt{2}$ is the root of the polynomial $x^2 - 2 = 0$ and hence algebraic. On the other hand, it had been proved that $\pi$ (by Ferdinand von Lindemann in 1882) and $e$ (by Charles Hermite in 1873) are not algebraic, hence transcendental. In fact, although it is very difficult to prove that a given number is transcendental, it was proven that (by Georg Cantor in 1874) most of the real numbers are transcendental. Irrational algebraic numbers cannot be represented exactly but, it was shown that the predicates involving them can be answered exactly [Yap and Dubé, 1995; Mehlhorn and Näher, 1995; Ouchi, 1997; Karamcheti et al., 1999; Mehlhorn and Schirra, 2001].

In StreetMaker, CGAL is heavily utilized for geometric computing. For 2D segment arrangements and other generic algorithm implementations we used the exact constructions with exact predicates kernel which is based on the GNU Multiple Precision Arithmetic Library (GMP)[7]. For the pedestrian network generation (*Chapter* 6), Bézier arrangements were used with the CORE library [Yu et al., 2010] which supports exact predicates for algebraic numbers[8].

Finally, interested reader may refer to Schirra [1997] and Li et al. [2005] which are very good survey papers on the topic.

## 5.6 Generic Algorithms

A set of generic algorithms are implemented within the core functionality of StreetMaker. Any application developed using StreetMaker can utilize these algorithms. The integrated algorithms and sample outputs are displayed in the following sections.

### 5.6.1 Delaunay Triangulation and Alpha Shapes

The theory of Delaunay triangulations and $\alpha$-shapes are discussed in *Sections* 4.1 and 4.2 respectively. Delaunay triangulations of point sets are implemented using a hybrid approach. First the input point set is divided into separate groups and then triangulated using the incremental construction algorithm shipped with CGAL triangulation package. Then, separately obtained triangulations are merged with the algorithm introduced by

[7] https://gmplib.org/

[8] In fact, we used quadratic Bézier curves which are represented by parametric quadratic polynomials. The roots of these polynomials might be irrational but they are algebraic. Therefore to support exactness, we needed a library which supports exact predicates for algebraic numbers.

Lee and Schachter [1980]. The steps of this merging algorithm is described in details with an example in *Appendix* B.2. Furthermore, if it is required (generally for big datasets), each subset of the input point set can be processed in parallel.

Recall that $\alpha$-shape of a point set is obtained by first calculating the Delaunay triangulation and then eliminating some of the simplices of the triangulation depending on the value of the $\alpha$. For computing the $\alpha$ shapes CGAL's 2D Alpha Shapes package is utilized. The computed Delaunay triangulations are supplied to this package with a selected $\alpha$ value to obtain the resultant $\alpha$-complex.

*Figure* 5.9 displays a synthetic point cloud in which three objects can be recognizable. *Figure* 5.10 displays the computed Delaunay triangulation for this point set from which the $\alpha$-complex is computed. Then, the connected sets of $\alpha$-edges that form polygonal regions are found. These polygons can either be imported into a PPVL (*Figure* 5.11) or SPPVL (*Figure* 5.12).

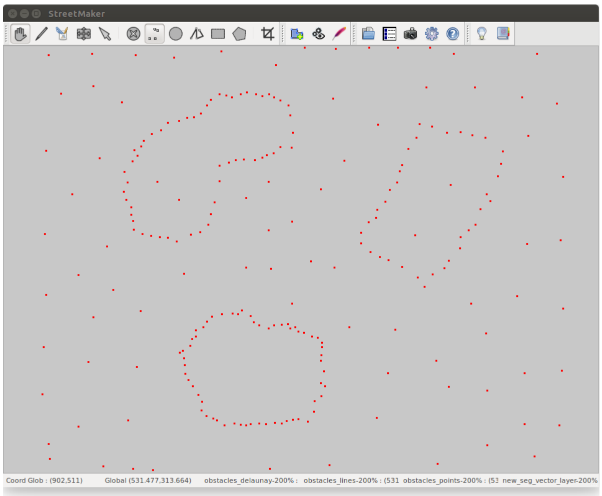

StreetMaker
Coord Glob : (902,511)
Global (531.477,313.664)
obstacles_delaunay-200% :
obstacles_lines-200% : (531
obstacles_points-200% : (53
new_seg_vector_layer-200%

**Figure 5.9:** A 2D point cloud in which three objects are recognizable.

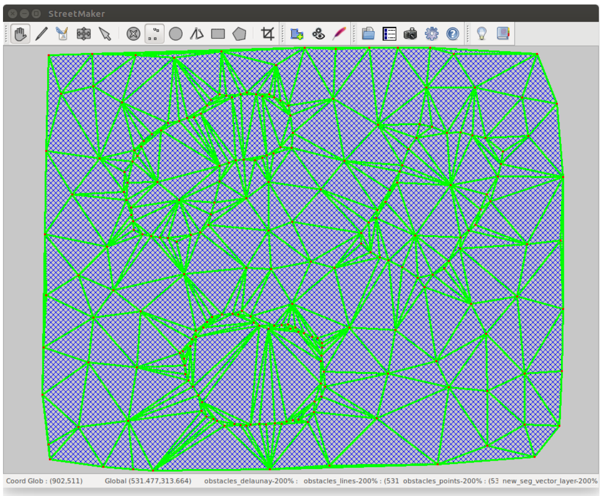

StreetMaker
Coord Glob : (902,511)
Global (531.477,313.664)
obstacles_delaunay-200% :
obstacles_lines-200% : (531
obstacles_points-200% : (53
new_seg_vector_layer-200%

**Figure 5.10:** Delaunay triangulation computed for the point set displayed in *Figure* 5.9.

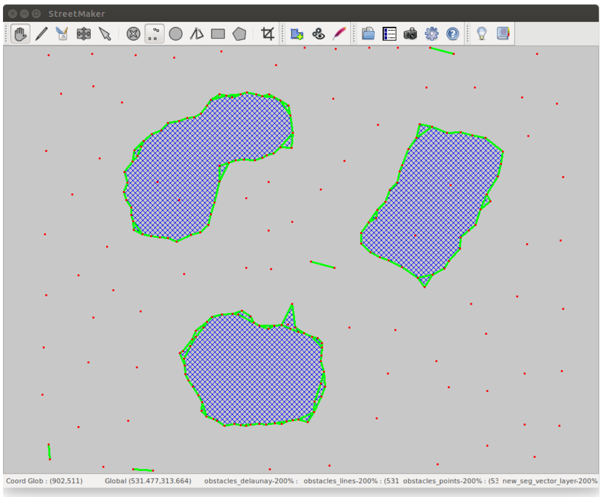

StreetMaker
Coord Glob : (902,511)
Global (531.477,313.664)
obstacles_delaunay-200% :
obstacles_lines-200% : (531
obstacles_points-200% : (53
new_seg_vector_layer-200%

**Figure 5.11:** $\alpha$-shape computed for the point set displayed in *Figure* 5.9 and converted into a PPVL.

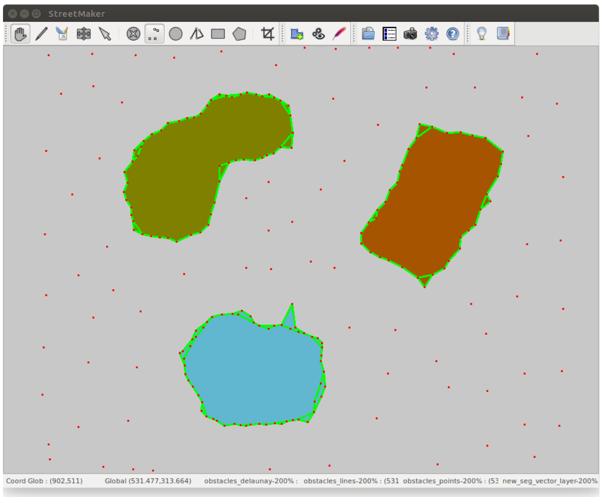

StreetMaker
Coord Glob : (902,511)
Global (531.477,313.664)
obstacles_delaunay-200% :
obstacles_lines-200% : (531
obstacles_points-200% : (53
new_seg_vector_layer-200%

**Figure 5.12:** $\alpha$-shape computed for the point set displayed in *Figure* 5.9 and converted into a SPPVL.

### 5.6.2 Medial Axis

Theoretical background for medial axis is discussed in *Section* 4.3.1. In StreetMaker, Segment Delaunay Graph (SDG) and polygon packages from CGAL [Karavelas, 2004] are used to compute the medial axis for polygons and polygon with holes. The procedure for computing the medial axis is as follows:

- The operator selects the objects for which the medial axis is going to be computed. At least one object must be selected in an activated SPPVL.
- The selected object geometries are merged together to obtain a set of polygon with holes ($S_{pwh}$) in which each polygon with hole represents a topologically connected component of the merge.
- SDG algorithm is run on the elements of $S_{pwh}$ one by one and Segment Voronoi Diagram (SVD) is obtained from SDG. Note that SVD is the dual of the SDG.

- The edges of SVD that lie outside the polygon with holes are eliminated. The remaining structure is the medial axis of the corresponding polygon with holes.

- The computed medial axis can be stored separately or distributed to semantic objects. That is, the intersections between the medial axis and the individual objects are computed and each object stores the part of the medial axis that lies on it.

*Figures* 5.13, 5.14 and 5.15 display examples of medial axes generated by StreetMaker.

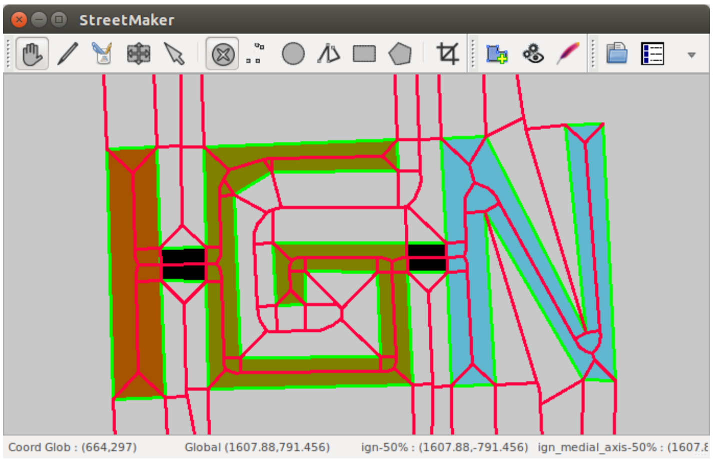


**Figure 5.13:** SVD of four semantic objects generated by StreetMaker (only the finite edges are displayed). Medial axis is a subset of the SVD.

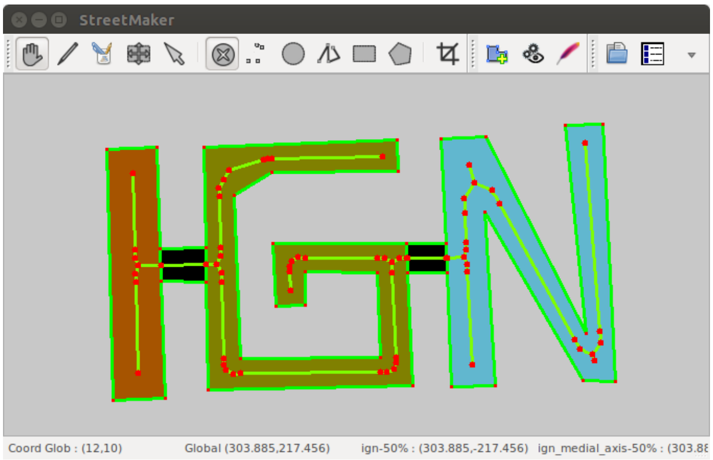


**Figure 5.14:** Extracted centrelines from SVD are distributed to the semantic objects.

### 5.6.3 Straight Skeleton

The theory of straight skeletons is discussed in *Section* 4.3.2. In StreetMaker, CGAL Polygon package is used to calculate the straight skeleton for polygons and polygon with holes. The procedure of computing straight skeletons is very similar to the one described for medial axis in *Section* 5.6.2. Only the SDG algorithm is replaced with the CGAL's straight skeleton algorithm.

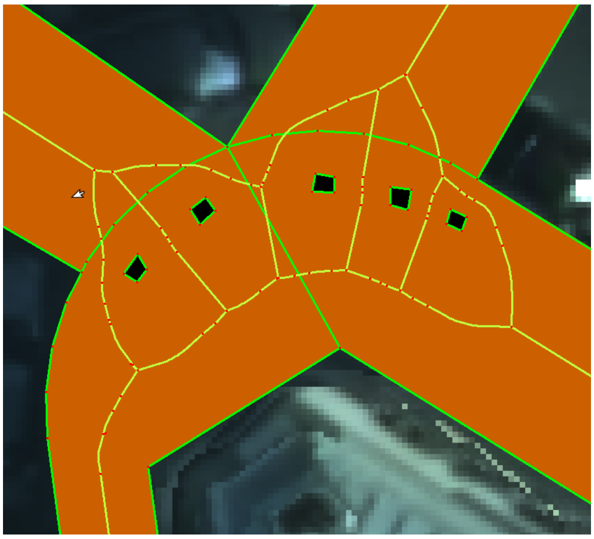
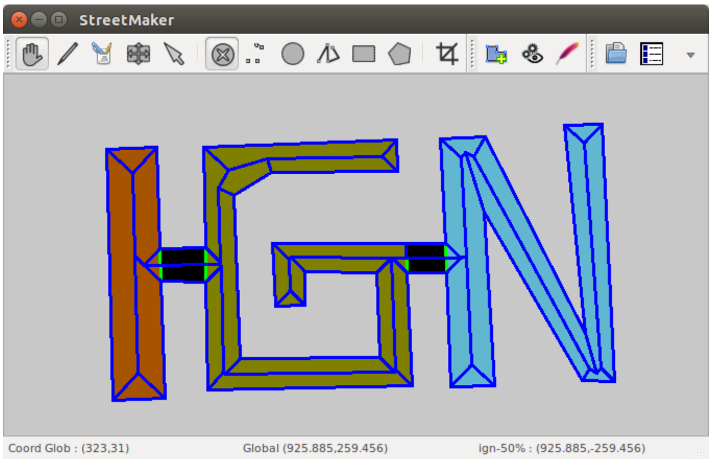


**Figure 5.15:** Medial axis centrelines for a real physical surface with obstacles (objects rendered in black are obstacles).

*Figure* 5.16 displays an example straight skeleton on the main window of StreetMaker. The non-skeleton edges (the edges that are incident to a vertex on the boundary of the input polygon) can be eliminated to obtain skeleton centrelines (*Figure* 5.17).



Coord Glob : (323,31)   Global (925.885,259.456)   ign-50% : (925.885,-259.456)

**Figure 5.16:** Straight skeleton generated by StreetMaker. See also *Figures* 5.13 and 5.14 for the SVD and medial axis of the same polygon.

## 5.6.4 Connectivity Graph

Connectivity graphs are discussed in *Section* 4.4. *Figure* 5.18 displays a connectivity graph for a road network. Each border edge of a semantic object is represented with a graph node and if the border between two semantic object is allowed to be crossed by an entity navigating on the surface of the objects, then the corresponding nodes are connected in the connectivity graph.

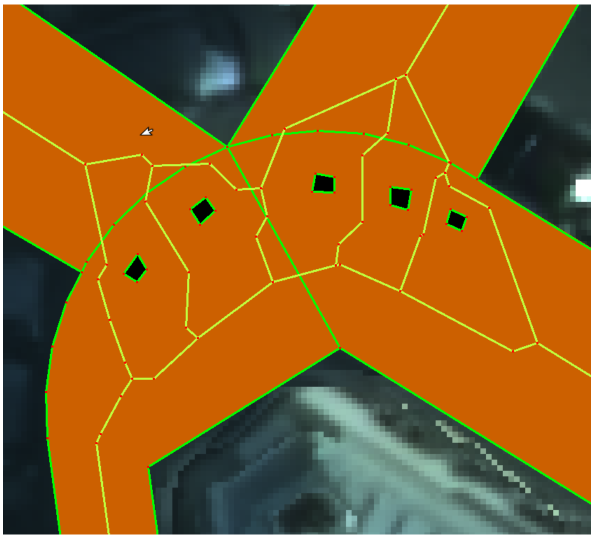


**Figure 5.17:** Straight skeleton centrelines for a real physical surface with obstacles (objects rendered in black are obstacles).

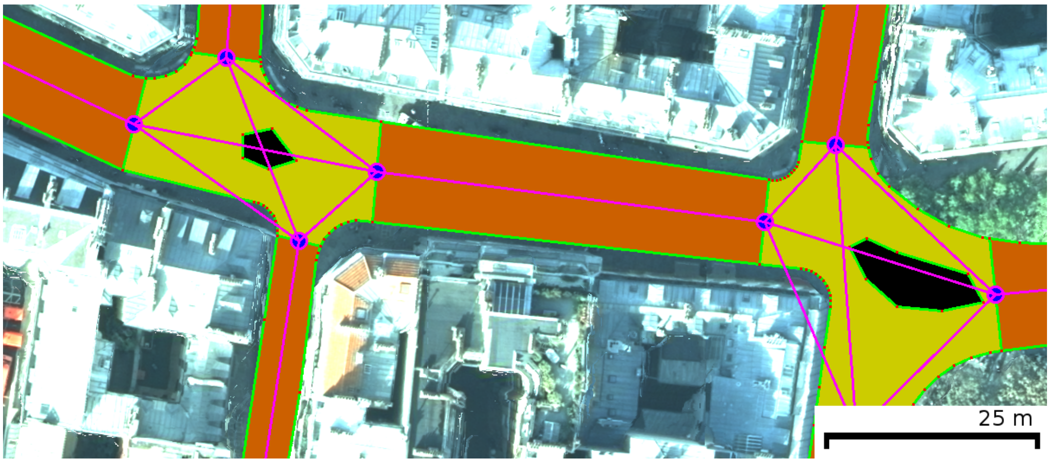


**Figure 5.18:** A connectivity graph for a road network generated by StreetMaker.

## 5.7 Implementation Details

StreetMaker has been implemented in C++ programming language with the utilization of various cross platform software libraries (C++ standard library for algorithms and containers; boost for smart pointers, unordered maps, object serialisation and graphs; wxWidgets and Gilviewer for GUI; CGAL for 2D arrangements and geometric computing; Geospatial Data Abstraction Library (GDAL) for processing vector data; Proj4 for coordinate transformations and pugixml for XML processing). The core StreetMaker is composed of about 30000 lines of code. And an application of pedestrian network generation (*Chapter* 6) that is built upon the core functionality is about 2800 lines of code.

Applications are developed on top of the core StreetMaker functionality by simply deriving a couple of classes. These classes are then used to specialize the GUI interface, object creation (generally new types of objects are defined for each application) and algorithms.

## 5.8 Conclusion

In this chapter, a framework called StreetMaker has been introduced. StreetMaker is designed for easy GIS application development especially for applications that require geometric computing. In fact, StreetMaker is a GIS by itself with its core functionalities without any further development. On the other hand, this core functionality can be used to built up specific applications that utilize the StreetMaker's data model and generic algorithms.

Thanks to the 2D arrangements data structure, as soon as data is inserted into a vector layer, the intersections are automatically computed and the underlying space is partitioned. Furthermore, by keeping the explicit topological relationships between the geometric primitives, geometric computation with StreetMaker is efficient and effective. And with the applied exact computation paradigm, calculations are exact and the algorithms are robust.

The three components of the core StreetMaker can further be improved as a future work. Firstly, more smart plotting tools can be developed for the basic GIS capabilities. For instance, edges can be detected in the images with sub-pixel precision and operator drawn lines can be snapped to previously detected edges. Secondly, the current data model can represent only 2D data which can be upgraded to 2.5D by adding the height information to each vertex of the 2D arrangements. Finally, new generic algorithms can be integrated into the framework. As the number of integrated generic algorithms increases, the application development process will get more and more easier. Therefore, when developing specific applications, a good strategy will be the isolation of the generic parts from the application specific algorithms and integrating them into the core generic functions of the framework. As a result, generic functions will be accumulated as the framework is used for different applications.

Developing a GIS from scratch is not an easy task for a limited number of developers with a limited amount of time. A good alternative would be using an open source GIS (e.g. QGIS or GRASS GIS) which would have a learning curve but on the other hand it could save the implementation time that had been spent for the GUI and the basic GIS capabilities development. One should have enough experience on both paths before deciding on the better approach. To be honest, it was not considered deeply whether it is better to start with an open source GIS or not. Our primary focus was on the development of the data model and the pedestrian network generation algorithm, and for practical reasons (e.g. having a nice GUI library, starting to implement as early as possible, license issues - not sure at the time of starting whether it will be an proprietary tool or open source), we chose to develop StreetMaker.

# Chapter 6

# Pedestrian Network Generation

## 6.1 Introduction and State of the Art

Starting from the 90s, personnel mobile computing devices such as smart phones and tablets have been changing our lives. These devices are capable of accessing wireless internet and satellite-based positioning systems, utilize a variety of location based services such as navigation, social networks, real time traffic information, emergency services, entertainment, advertising, etc. [Steiniger et al., 2006]. Especially, car navigation systems are very successful and may be the most common location based service which provide vehicle's current location as well as routes to the selected destinations.

Digital map databases are at the heart of a navigation system. Great amount of data have been collected throughout the years and road network databases are now well developed and widely available for many countries. After the success of car navigation systems, pedestrian navigation systems also have been getting interest. Especially, disabled individuals would benefit a lot from a pedestrian assisting navigation system. However, the data for pedestrian routes is far from being sufficient to generate digital map databases. The lack of data problem has been tried to be solved by using the vehicle map databases for also serving the pedestrians. For instance, Google Maps delivers walking routes based on road networks with the disclaimer: "the route may be missing sidewalks or pedestrian paths."

Despite the efforts of utilizing vehicle networks as a basis of pedestrian navigation systems, it has been demonstrated that this approach is restricted and it has little benefit. Chin et al. [2008] compared the vehicle networks and pedestrian networks qualitatively based on a walk-ability index and concluded that the nature of pedestrian and vehicle routes are quite different. Unlike motorized vehicles, pedestrian movement takes place along pedestrian paths (sidewalks, parks, pedestrian crossings, pedestrian bridges, etc.), not along the street lanes and are not constrained by the boundaries of the road. Pedestrians face different challenges such as movement with a higher degree of freedom [Holone et al., 2007; Stark et al., 2007; Gaisbauer and Frank, 2008]. As a result, pedestrian networks and the vehicle networks are considerably different and new approaches are required for generating pedestrian networks.

In general, pedestrian networks usually require much finer resolution than the road networks due to the unrestricted pedestrian movements. However, generating very detailed pedestrian networks might be very costly. As a result, pedestrian networks are generally generated at different levels of details depending on the needs of the applications. For instance, in urban planning and unaided pedestrian navigations systems, relatively rough networks are acceptable [Walter et al., 2006; Elias, 2007; Ballester et al., 2011]. On the other hand, more detailed networks are necessary for mobility impaired pedestrian and autonomous vehicle navigations [Helal et al., 2001; Beale et al., 2006; Mayerhofer et al., 2008; Kasemsuppakorn and Karimi, 2009].

Ballester et al. [2011] generated a rough pedestrian network using road networks and building footprints. Their method is based on a buffering approach with the following assumptions: there exists a sidewalk between every building and road and there exists a pedestrian crossing at every intersection. They ignored the pedestrian walkways that are not adjacent to road networks such as parks or other types of pedestrian zones. Besides, the assumptions might not be valid at all places. Walter et al. [2006] first created binary raster maps by marking the walkable surfaces manually on raster data, then running a morphological operator to find out the skeleton of the walkable surfaces. After smoothing the obtained skeleton a rough pedestrian network is generated. Elias [2007] discussed the methods and difficulties of generating a tailored geodatabase for pedestrian routing. She used the ground floor plans of public or industrial buildings (e.g. train station) to generate the indoor routes by using an approximated medial axis. In addition, outdoor pedestrian routes are estimated from the digital cadastral maps and a topographic database. All these different data sources were merged together and compatibility issues were solved via commercial GIS. Furthermore, interest points such as building doors were connected to the generated network without considering the geometry and topology of the network, therefore sometimes resulted in routes passing through the buildings.

In urban planning, a common objective is to make cities more pedestrian-friendly helping to increase the physical activity of its inhabitants and at the same time decrease traffic congestion and pollution. Researches need pedestrian networks in order to audit pedestrian environment and evaluate pedestrian network connectivity [Southworth, 2005; Clifton et al., 2007; Chin et al., 2008]. Hence, the priority in urban planning is the topological correctness of the pedestrians networks and geometrically rough networks are acceptable.

Helal et al. [2001] presented a wireless navigation system called Drishti for visually and mobility impaired pedestrians. They integrated several technologies including wearable computers, voice recognition and synthesis, wireless networks, GIS and Global Positioning System (GPS). Drishti constantly guides the blind user to navigate based on static and dynamic data. The static data is formed by centrelines of the walkways which was created manually for the study of area (the University of Florida (UF) campus) and the dynamic data comes from the integrated sensors.

Beale et al. [2006] constructed a spatial database containing the pedestrian route network and barriers that blockade the navigation. The research was undertaken in the town centre of Northampton, UK (approx. $2km^2$). They performed a questionnaire among the wheelchair users to identify urban barriers with their weighted effect to navigation. Guided by the results of this questionnaire, they generated a very detailed barrier map via field surveys. Moreover, they also integrated attributes such as the name of the streets and metrics such as slope information (automatically extracted from DEM), surface type, and surface quality into the database. The pavement centrelines manually digitized with the assistance of aerial photos as backdrops. Finally, a GIS application running the model, calculated the routes through the pedestrian route network that take account of barriers to accessibility.

Mayerhofer et al. [2008]; Kasemsuppakorn and Karimi [2009] also developed comparable systems for visually impaired and wheelchair users respectively similar to Helal et al. [2001] and Beale et al. [2006]. In all of these works, the walkway centrelines are generated manually using a GIS. Moreover, Mayerhofer et al. [2008] expressed the need for a semi-automatic or automatic tool for generating and maintaining the created pedestrian networks. The author would like to quote mentioned sentences: "The level of detail has to meet the requirements for an exact description of the environment. Another demand is the up-to-datedness of the digital map. This is a challenging task because of the required manpower. Therefore, a future goal is to develop software for semi-automatic or even automatic generation of navigable maps out of cadastral surveying or any similar available data." Furthermore, Beale et al. [2006] also commented on the automatic generation of pedestrian routes: "Due to the large range of features that may define the pavement edge (e.g. building outlines, walls, road edges), development of a rigorous automated method for route identification would have been extremely difficult. Instead, the pavement centrelines were manually digitized using the Ordnance Survey Land-line data as a backdrop and incorporating local knowledge of favoured pedestrian routes (e.g. passages between buildings or through shopping arcades). Large-scale aerial photography and ground-truth surveys were also used to validate the routes." Finally, Kasemsuppakorn [2011] remarked: "Currently, researchers requiring pedestrian networks for studies generate their own data. One major problem with this approach is that the produced data is very specific and only useful for a particular scenario and chosen area. This and other observations indicate that there is an absolute need for developing new methodologies and techniques for acquiring and maintaining pedestrian networks; this is an important area of study for further advancements in pedestrian-centric location based applications."

We propose a pedestrian network generation process based on our framework which is described in *Chapter* 5 and whose theoretical background is defined in *Chapters* 2, 3 and 4 [Yirci et al., 2013]. Our claim is that the offered model might be a good candidate for the researcher needs mentioned in the previous paragraph. The output of our process is a static obstacle avoiding pedestrian network graph which is guaranteed to be geometrically exact. Furthermore, the generated network graph not only includes the centrelines but also it can store areal information inherited from an hierarchical object model which capture the space occupancy of the urban objects on the area of study. The advantages, limitations, usages and prospects of our network generation process has been discussed within enlightenment of the state of the art pictured in this *Section* (6.1). Before describing our model, the author would like to mention two more aspects of the problem related to information models and evaluating the quality of the pedestrian networks.

The content of a map and techniques to realize a map are two different but closely related topics in which the former guides the latter. Beale et al. [2006], and Kasemsuppakorn and Karimi [2009] performed field studies and questionnaires to find out the needs of pedestrians with wheelchair. Moreover, Laakso et al. [2013] presented a study focusing on the information content of the geospatial databases used to guide pedestrians. Recently Neis and Zielstra [2014] provided a brief survey on the information content of pedestrian networks. Besides these works, Stark et al. [2007] discussed the different possible representations of pedestrian networks to the users of a navigation system. Our work is neither related to the representation nor information model of a pedestrian network but creation and maintenance of such a constrained data model.

Wiedemann [2003] proposed a method for evaluating the quality of automatically extracted road networks through image processing. Karimi and Kasemsuppakorn [2013] adapted this method to evaluate the pedestrian networks. The evaluation method consists of two steps: (i) comparing the ground-truth with the generated pedestrian network through a matching process, (ii) calculating the quality measures of four evaluation criteria (geometric completeness, geometric correctness, topological completeness and topological correctness). Karimi and Kasemsuppakorn [2013] compared three different pedestrian networks with respect to a pedestrian network baseline as a ground-truth. However, their ground-truth was also an approximation to the actual pedestrian network. On the other hand, our method computes geometrically and topologically exact pedestrian networks which can be used as a ground-truth.

Apart from pedestrian network graphs produced in GISs, similar graphs have been produced in computer graphics (for applications such as virtual reality, video games, crowd animation, etc.) and robotics for path planning and environment representation. Lamarche [2009] outlined two main approaches used in these fields: roadmap and cell decomposition methods. Roadmaps are graph-based networks and different approaches have been used to compute them. There exist visibility graph methods [Lozano-Pérez and Wesley, 1979; Arikan et al., 2001], Voronoi-based methods [Hoff et al., 1999; Bhattacharya and Gavrilova, 2008; Geraerts, 2010] and Probabilistic Roadmap Methods (PRMs) [Kavraki et al., 1996; Svestka and Overmars, 1998; Nieuwenhuisen et al., 2007]. Cell decomposition methods subdivide the space into convex cells and represent the resulting navigable environment with maps called navigation meshes. Then, path planning is performed with a graph search algorithm running on the mesh structure. Navigation meshes initially designed for path planning in 2D planar surfaces (embedded in 3D) [Snook, 2000], later versions for multi-layered and non-planar environments have also been developed to address 3D scenes [Lamarche, 2009; Jorgensen and Lamarche, 2011; Saupin et al., 2013]. Refer to Kallmann and Kapadia [2014] for a survey of the existing methods for navigation meshes.

According to the techniques (roadmaps and cell-decomposition) introduced in the previous paragraph our work can be classified as a Voronoi-based roadmap approach. When compared to the roadmap approaches, our pedestrian network graphs are generated relatively for large-scale regions, geo-referenced and further enriched: structured with additional nodes - border nodes, parsed into sub-graphs - trajectory arcs with optimum path clearance to obstacles (exact minimum distance to obstacles for each trajectory arc) and exact total length, point projections onto the graphs (entry points to the graphs) and additional semantic information captured from the source semantic planar partitions. Moreover, we do not address path planning but static graph generation for supporting path/motion planning applications. On the other hand, computer graphics and robotics applications generally aim to generate collision-free dynamic graphs (environments with mobile obstacles, e.g. other moving entities) for multiple navigating agents. Furthermore, the computed paths that directly follow our graphs will not be optimal in terms of shortest paths which is sometimes among the goals of robotics applications.

Our network graphs depend on semantic planar partitions. Instead of computing the Voronoi-based roadmaps, cell-decomposition approach can also be applied. In this case, the semantic planar partition can be triangulated and the resulting structure will be a semantic navigation mesh. Then, the techniques already developed for path finding in navigation meshes can be applied [Kallmann, 2010] with possible improvements via utilizing the captured semantic information from the ground surfaces.

## 6.2 Pedestrian Network Graph

A pedestrian network graph can be formally defined by a planar graph $G = (V, E)$ embedded either on a plane in $2D$ or on the surface of an ellipsoid in $3D$ which represents Earth. Any vertex $v_i \in V$ is associated with a coordinate pair or triple defining a point location on Earth in a projected or geodetic coordinate system. The edges $e_i \in E$ are representing the centrelines of the walkways. In theory, the centreline of a walkway is the medial axis of the shape defined by the walkway and it can be any plane curve in 2D or space curve in 3D. In practice, calculating such a general curve might be extremely difficult. Moreover, such a calculation requires the mathematical definition of the shapes which is almost impossible to get for real life objects, hence an approximation is needed. In fact, in GIScience, the shapes of the areal objects are approximated by polygons possibly with holes. In this work, the centrelines of the walkways are computed exactly from the polygon with holes using the medial axis transformation (*Section* 4.3.1 and *Section* 5.6.2) with the exact computation paradigm (*Section* 5.5). As a result, the generated pedestrian networks are exact and only limited by the approximation of the shapes.

The input to the pedestrian network generation process is a hierarchical semantic planar partition which might be composed of areal and punctual objects. The areal objects represent the walkways and the punctual objects may represent some other interest points such as doors of the buildings, metro stations, bus stops, etc. The output of the process is a 2D planar graph $G = (V, E)$. The edges and vertices/nodes of $G$ are classified and grouped in order to structure the obtained pedestrian network. The edges of $G$ (denoted by $G(E)$) are grouped under the formations called trajectory arcs and the nodes of $G$ (denoted by $G(V)$) are classified into two main groups as trajectory nodes and ordinary nodes.

***Trajectory arcs:*** A trajectory arc is a connected sub-graph of $G$ which represents an homogeneous part of $G$. That is, the character of the walkway does not change along a trajectory arc.

***Trajectory nodes:*** The graph nodes that determine the start and end points of the trajectory arcs are called trajectory nodes. Trajectory nodes further classified into three sub-groups: start/end nodes, junction nodes and border nodes. The degree of a node is defined by the number of incident edges to the node. Degree-one nodes represent the start and end points of the graph. Similarly, the nodes with degree greater than two are the junction points of multiple paths on the graph. Finally, border nodes identify intersection points between the centrelines (i.e. edges of the graph) and borderlines. Borderlines represent the boundaries between the areal objects.

***Ordinary nodes:*** The graph nodes other than the trajectory nodes are named as ordinary nodes and all of these nodes are of degree two.

Note that, trajectory arcs can be either open or closed. An open trajectory arc has two distinct trajectory nodes whereas a closed trajectory arc possesses only one trajectory node which is both the start and end point of it. Notice also that junction and border trajectory nodes are shared between different trajectory arcs.

Since the centrelines of pedestrian networks are generated using medial axis of polygon with holes, the edges of the graph are either line or parabola segments. Moreover, the straight skeleton operator (*Sections* 4.3.2 and 5.6.3) can also be used for centreline construction which only generates line segments but remember that on the contrary to the medial axis, the points on a straight skeleton are not equidistant to the boundary of the polygons when they are not convex.

One of the primary usages of pedestrian networks are the itinerary calculations for navigation. In a navigational query the total length of a possible route is one of the main concerns. To support this feature, the total length of each trajectory arc is calculated. Furthermore, the minimum width of the walkways may be crucial for pedestrians using wheelchairs or other means of devices as well as autonomous vehicles that may navigate on the network. Therefore, a minimum width is calculated for each trajectory arc. The minimum width is the maximum radius of a disk translating along the trajectory arc that is fully enclosed within the walkway.

Any valuable data that has been stored in the input planar partitions can also be embedded into the graphs. That is, any semantic information attached to an object can be reachable from a graph node. The semantic information of an object that is not directly linked to geometry can also be utilized through the object hierarchy. For instance, if there exists a parent object representing the whole street, the graph nodes lying on the surface of the street can notify a pedestrian navigating on the street as soon as she/he enters/exists the street. Furthermore, a visually impaired pedestrian can be informed when she/he is approaching an obstacle or barrier based on the input planar partition. Besides the semantic information, the captured 2D geometry of the areal objects can also be embedded into the graphs which may provide a more degree of freedom in making decisions for some smart applications.

## 6.3 Pedestrian Network Generation Process

As already mentioned in the previous *Section* (6.2), the input to the pedestrian network generation process is a hierarchical semantic partition composed of areal and punctual objects. *Figure* 6.1 displays a very simple semantic partition composed of two areal and four punctual objects. The rest of this section will be discussed along with this sample semantic partition.

The process starts with merging the polygon with holes that describe the boundary of the walkway objects. Therefore, the centrelines for the merged surfaces can be calculated. After the merging step, although it is logical to end up with a big single polygon with holes, it is not mandatory. That is, disconnected areal objects are allowed in which case the union off all object surfaces is a set of disconnected polygon with holes ($S_{pwh}$). *Figure* 6.2 displays the union of the two areal object surfaces.

In the second step, centrelines of the walkways are generated. That is, the medial axis (or straight skeleton) of each polygon with holes in $S_{pwh}$ is computed. Note that, if the cardinality of $S_{pwh}$ is greater than one, then the generated pedestrian network graph will be disconnected. *Figure* 6.3 displays the computed medial axis for our example.

After generating the medial axis the basic pedestrian network graph is obtained. The nodes of this basic graph is inspected and the ones with degree-one and degree greater

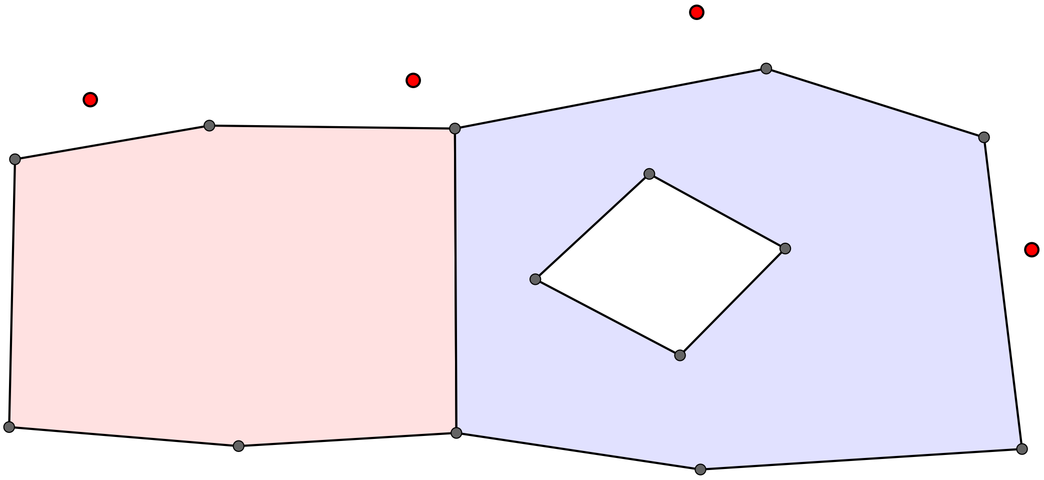


**Figure 6.1:** A simple semantic planar partition for pedestrian network generation which is composed of two areal and four punctual objects.

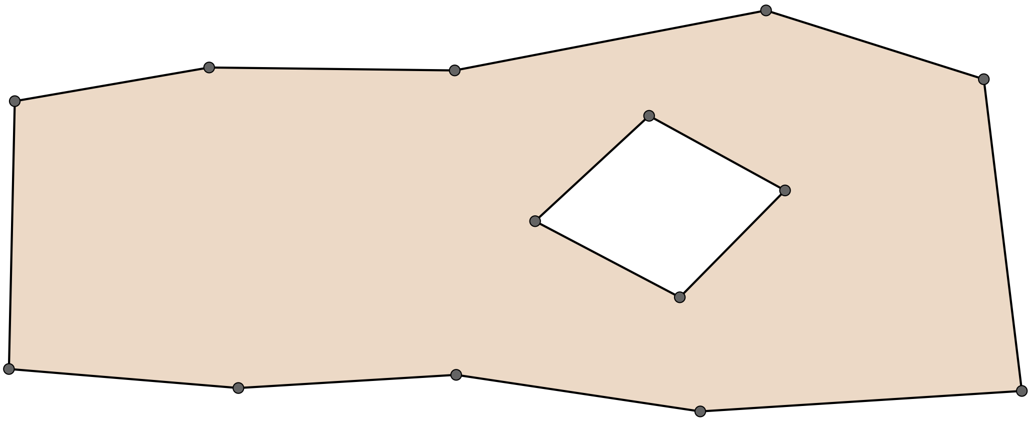


**Figure 6.2:** The surfaces of the areal objects are merged into a polygon with holes.

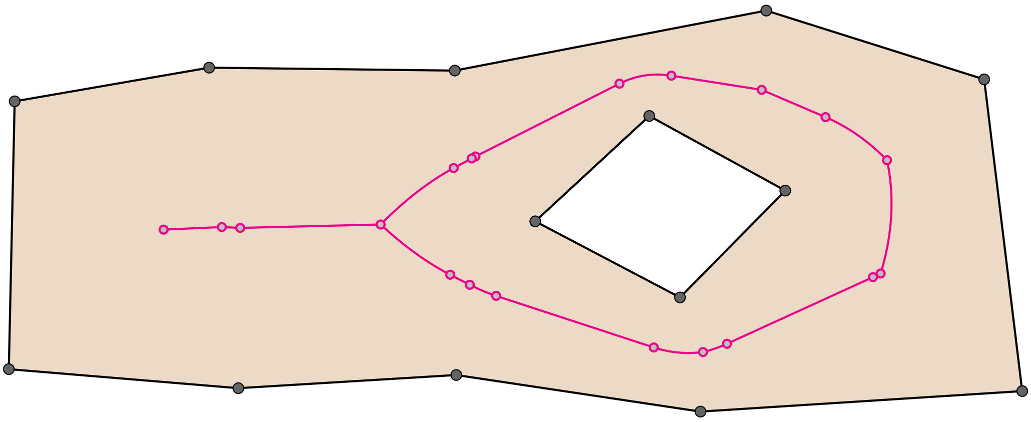


**Figure 6.3:** Centrelines of a polygon with holes generated with the medial axis transform.

than two are classified as trajectory nodes. However, the boundary trajectory nodes have not been classified yet. To accomplish this final sub-step, all borderlines within the semantic partition are detected using the topological relationships between the objects and their intersection points with the medial axis are computed. Then, these points are also added to the graph as boundary trajectory nodes (*Figure* 6.4).

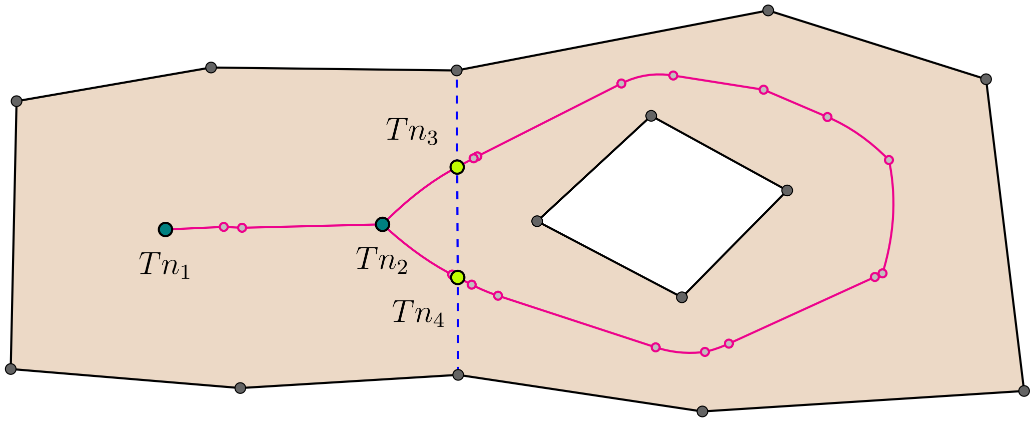


**Figure 6.4:** Trajectory nodes: $Tn_1$ is an entry/exit node, $Tn_2$ is a junction for multiple trajectory arcs, $Tn_3$ and $Tn_4$ are boundary trajectory nodes which are computed by intersecting the borderlines with the centrelines.

The fourth step is related to the computation of trajectory arcs. Trajectory arcs are recursively searched within the graph and as soon as one of them is detected, its exact total length (*Figure* 6.5) and minimum width (minimum distance to the borders of the bounding objects when translating on the edges of the trajectory arc) is computed (*Figure* 6.6).

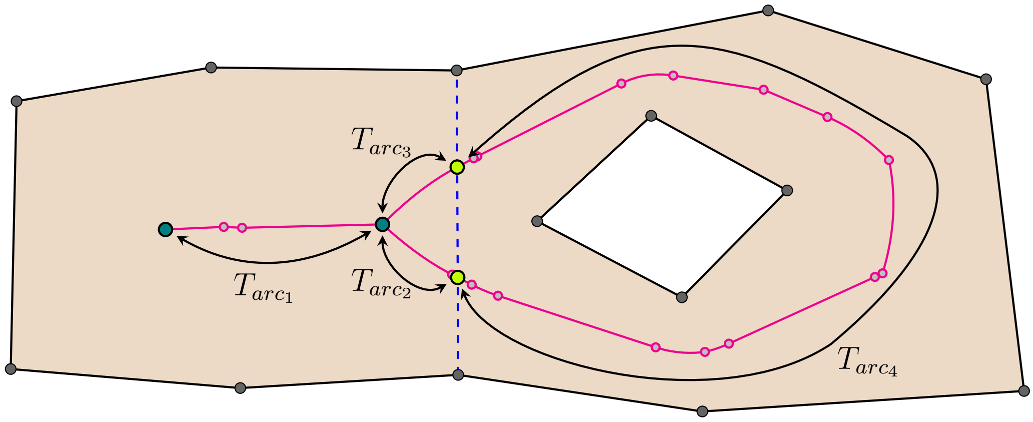


**Figure 6.5:** Trajectory arcs and their lengths are calculated: trajectory arcs are defined between two trajectory nodes (not necessarily distinct).

The final step of the process is related to the punctual objects which are projected on the closest edge of the pedestrian network graph. The projected point locations do not form a node on the graph but their positions are stored within the trajectory arcs. These projected points represent the entry and exit points of the graph (*Figure* 6.7). This completes the pedestrian network generation process. Please refer to *Section* 6.6 for the implementation details.

The next two sections demonstrate the operation of the pedestrian network generation process on the two separate semantic planar partitions. In these two applications, different

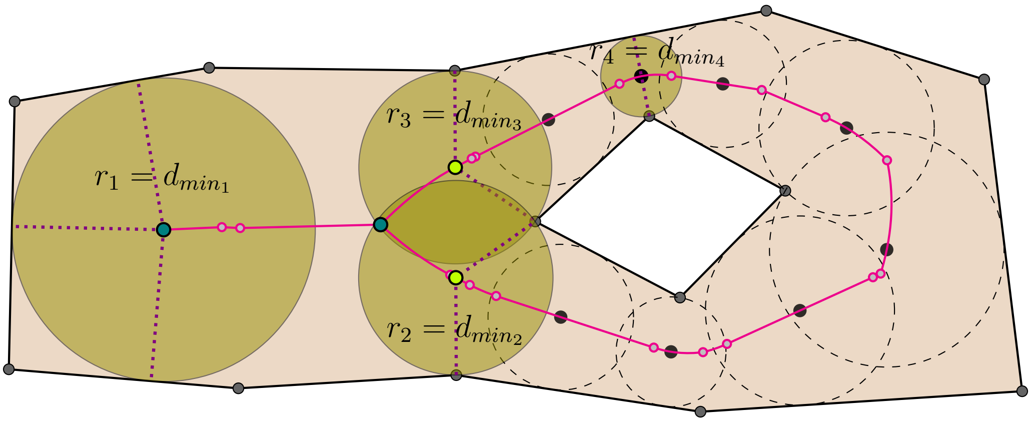


**Figure 6.6:** The minimum with of a trajectory arc is defined by the maximum radius of a disk translating along the trajectory arc that is fully enclosed within the corresponding polygon with holes.

data sources were utilized with different characteristics. The pipelines introduced in *Chapter* 1 for creating the semantic planar partitions are discussed in the following two *Sections* (6.4 and 6.5).

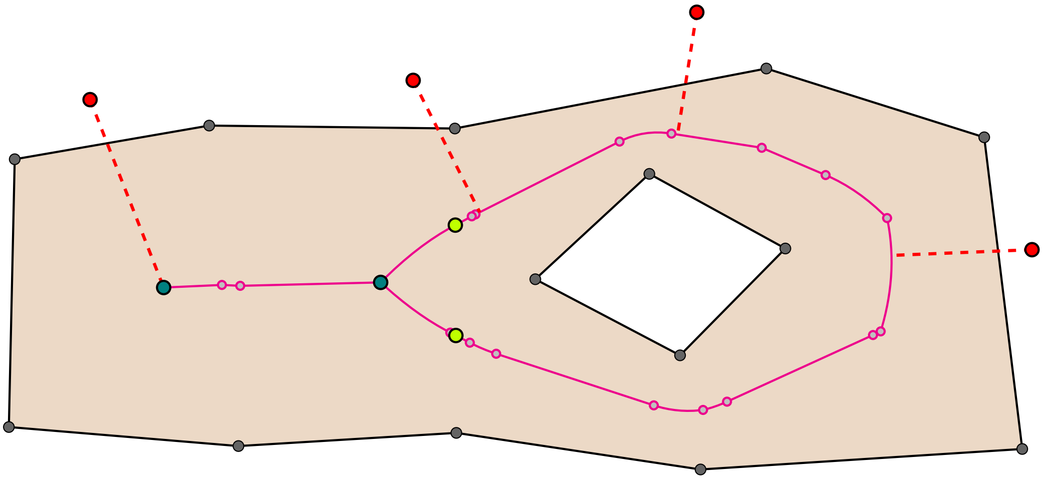


**Figure 6.7:** Projecting punctual objects onto the pedestrian network graph: punctual objects are projected onto the closest graph element (edge or node).

## 6.4 Application-1: Saint-Sulpice Paris VI

In this application a pedestrian network was generated for Saint-Sulpice region in Paris. The area of interest is about 56 hectares and composed of 57 building blocks. A building block is surrounded by sidewalks which is connected to other building blocks via pedestrian crossings or other types of walkways.

All applications start with the construction of a semantic planar partition over the region of interest. *Figure* 6.8 displays the constructed semantic planar partition for the Saint-Sulpice region.

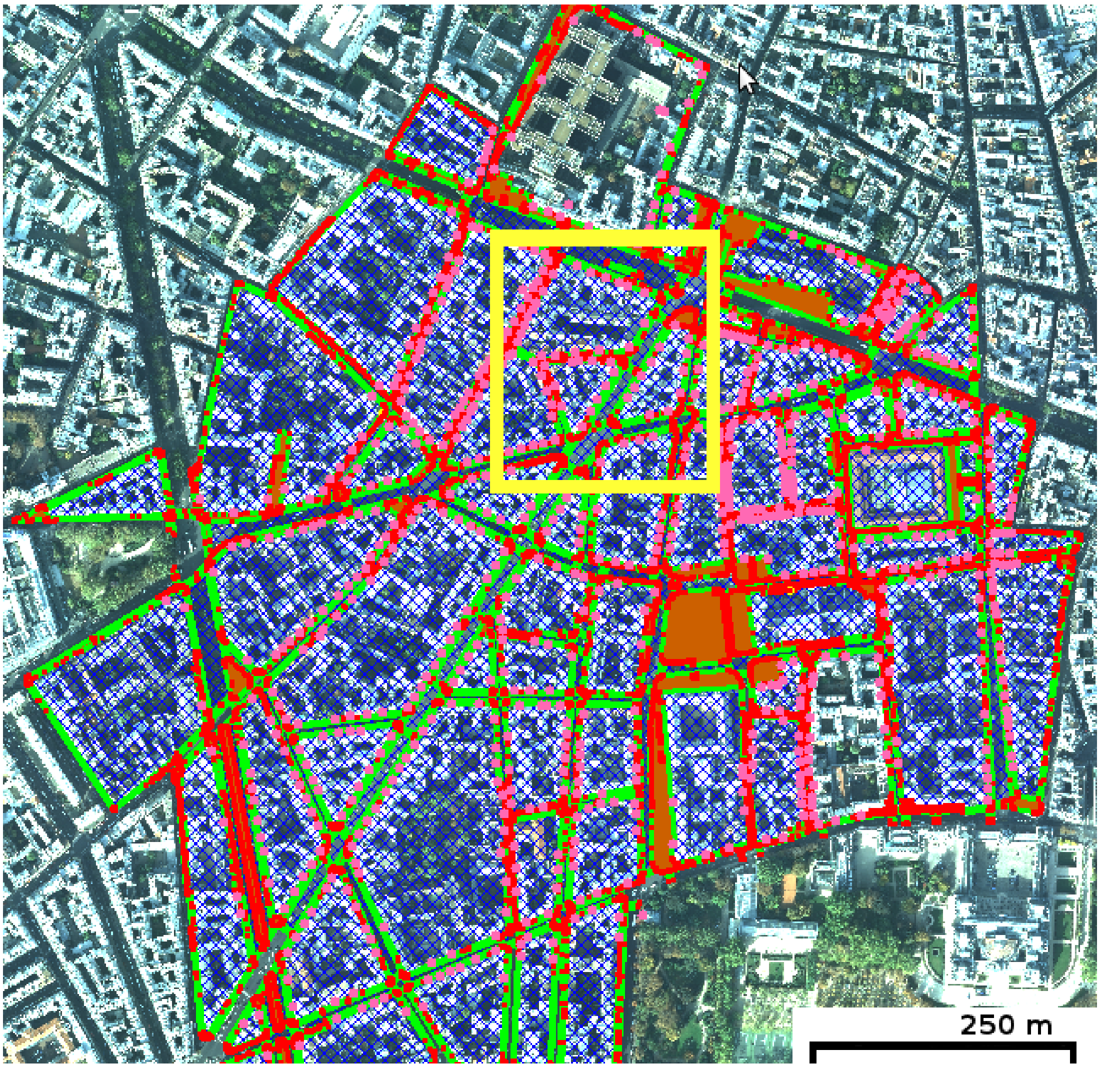


**Figure 6.8:** Saint-Sulpice region in Paris and its semantic partition: the pedestrian network graph computed for the highlighted region is displayed in the *Figure* 6.12

### 6.4.1 Creation of the Semantic Planar Partition

City of Paris has made a set of vector data public, called OpenData Paris[1]. For the initial planar partition we utilized four different datasets from OpenData Paris and an internal address data base from IGN called BD Adresse®[2]. The details of the input vector data are given in the *Table* 6.1. Observe that, the vector data used from OpenData Paris are linear, that is geometric features described in these datasets are composed of line segments connected to each other which do not necessarily form closed rings, i.e. polygons. Therefore, when the OpenData Paris vector datasets were converted to 2D arrangements and merged into a single vector layer, the resulting vector layer was far from being a valid planar partition. An ideal valid planar partition should correctly model the space occupancy / footprints of the geospatial urban objects. In such cases, we need to use the methods described in *Section* 5.4.2.

OpenData Paris vector datasets are not free of geometric / topological errors and missing

[1] http://opendata.paris.fr/page/home/

[2] http://professionnels.ign.fr/bdadresse

| **Data** | **Source** | **Content & Type** |
|---|---|---|
| Sidewalks | OpenData Paris | Outlines of the sidewalks, linear |
| Buildings | OpenData Paris | Outlines of the buildings, linear |
| Signalization | OpenData Paris | Outlines of the pedestrian crossings, lampposts, etc. |
| Poles | OpenData Paris | Outlines of the poles, linear |
| BD Adresse® | IGN | Locations of the building numbers corresponding to the entrance of the buildings, punctual |

**Table 6.1:** Employed datasets for creating the Saint-Sulpice semantic planar partition.

data. For example, when signalization and sidewalks vector data are merged, the end points of the pedestrian crossings do not lie exactly on the borders of the sidewalks in most of the cases (*Figures* 6.9, 6.10). Moreover, borders of some buildings and sidewalks are not complete. The simplification and error fixing tools introduced in *Section* 5.4.4 are used to fix many of these errors with the thresholds displayed in the *Table* 6.2. Then, manual corrections were done for completing the missing data and fixing the errors that could not be fixed by the automatic tools. The resulting planar partition (2D arrangement) is composed of 27877 vertices, 27561 edges and 4274 faces[3].

| **Tool** | **Threshold** | | |
|---|---|---|---|
| Redundant geometry removal | Degrees | → | 0.1 |
| Merging vertices within a neighbourhood | Radius | → | 5 cm |
| Auto connecting lines | Distance | → | 10 cm |

**Table 6.2:** Simplification and error fixing threshold values adjusted for the OpenData Paris vector dataset

The next step is the creation of semantic objects and the object hierarchy on top of the planar partition. Recall from *Section* 5.4.2 that semantic objects are created by selecting and grouping geometric primitives and previously created semantic objects. For this application, four different types of objects were defined: physical roads, logical roads, obstacles and street numbers.

***Physical roads:*** These objects represent physical surfaces that pedestrians can walk on, i.e. the walkways. Typical urban objects for this category are sidewalks, parks, pedestrian crossings, squares, etc. Physical roads lie at the lowest level of the object hierarchy since they correspond to the real physical surfaces.

***Logical roads:*** These objects are parent objects for the physical road objects, representing the logical formations on the city surfaces such as streets, avenues, etc.

***Obstacles:*** Any static object on walkways such as street furnitures, lampposts, poles and waste baskets are considered as obstacles.

***Street numbers:*** Street numbers correspond to address point locations. These are punctual objects indicating the exit / entrance of the buildings.

[3] Note that these numbers do not satisfy the Euler's formula (*Equation* 1 in *Chapter* 2) due to existing isolated vertices and unconnected components (remember that Euler's formula is valid for connected planar graphs).

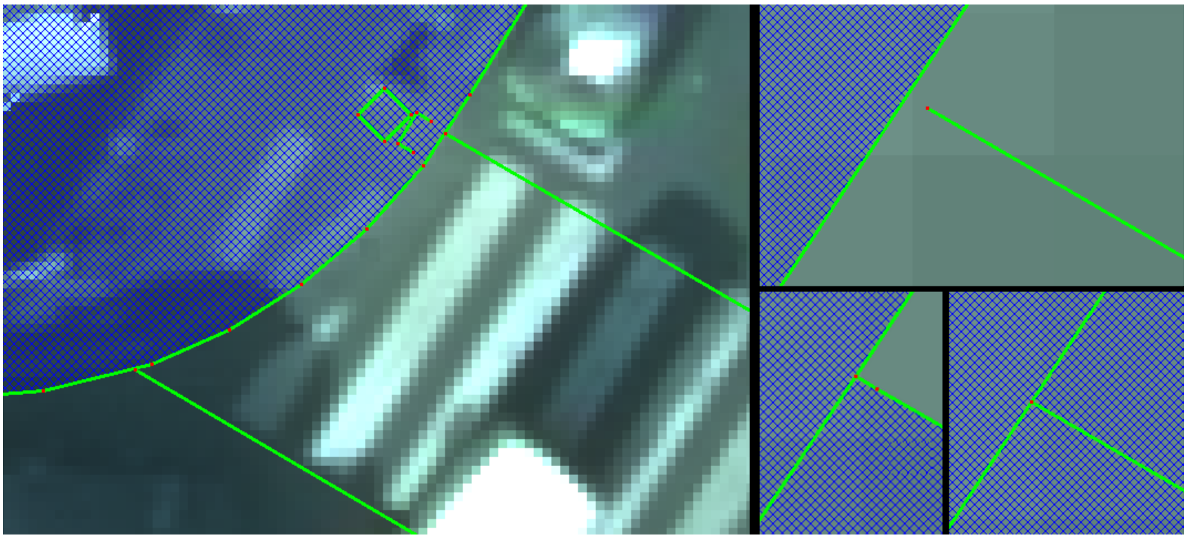


**Figure 6.9:** Auto connecting lines in OpenData Paris: an example of handling under-shoot or coordinate mismatch error. In the sub-figure on the left, pedestrian crossing outlines seem to touch with the outlines of the sidewalks, however when zoomed in, it can be seen that they are not (top right). Threshold based auto connecting tool is used to fix these errors by extending one the lines (bottom left) and removing extra geometry (bottom right).

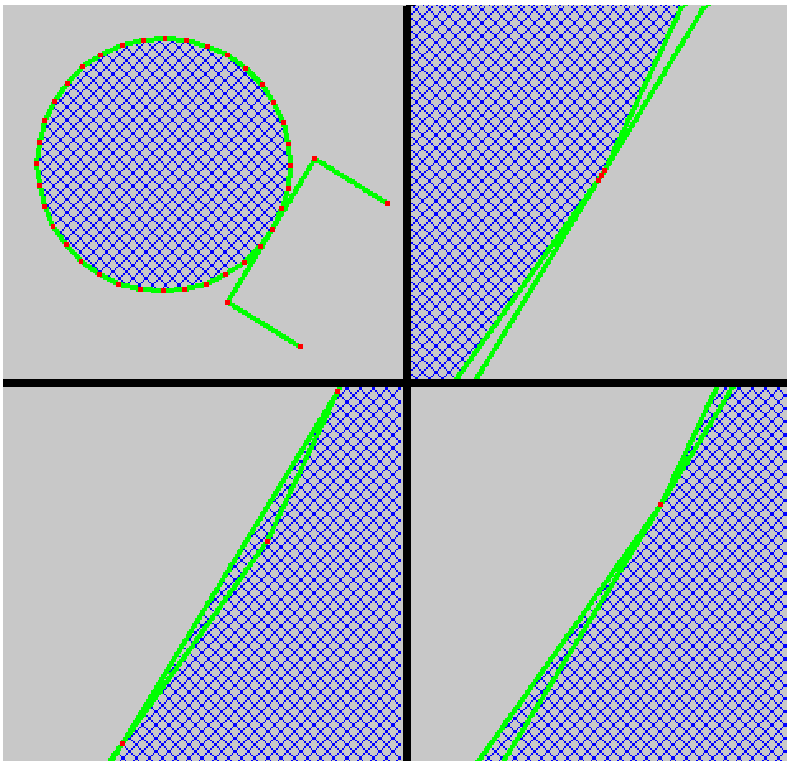


**Figure 6.10:** Merging vertices within a neighbourhood in OpenData Paris: in the top left sub-figure, it looks like a line is tangent to a circular polygon. In fact, when zoomed in (top right and bottom left sub-figures), it can be seen that the line intersects with two edges of the polygon forming a tiny triangle. Merging vertices within a neighbourhood tool is used to collapse triangles like this into a single vertex (bottom right).

The overall process for creating the semantic planar partition took about 5 hours of an operator which includes converting input datasets to 2D arrangements, running automatic error correction and simplification tools, manual error fixing and manually completing the missing data and creating the semantic objects. Then, the pedestrian network generation algorithm was run on the semantic partition which took about 50 minutes[4] to compute the pedestrian network using medial axis. *Figures* 6.11 and 6.12 display some parts of the generated pedestrian network graph with different levels of detail[5].

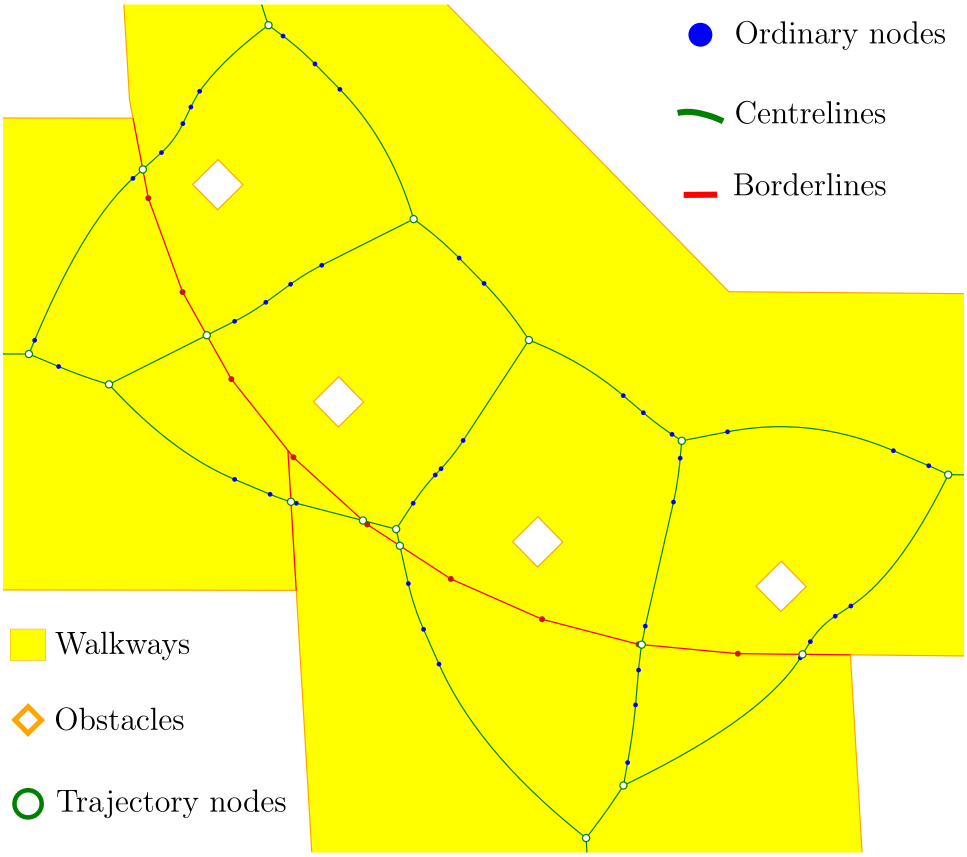


**Figure 6.11:** A detailed view of the generated pedestrian network for the Saint-Sulpice region in Paris. In the figure, a part of the pedestrian network graph around the intersection of two pedestrian crossings and a sidewalk is displayed. In addition, there exists four obstacles (poles) on the surface of the sidewalk.

## 6.5 Application-2: CASQY

The region of interest for this application was the urban conurbation of the Saint-Quentin-en-Yvelines (CASQY) in France. Pedestrian network graphs were generated for two separate regions (region-1 & region-2) in CASQY. The total surface area of the interested regions is 127 hectares (region-1: 36 ha, region-2: 91 ha). The input semantic planar partitions to the pedestrian network generation pipeline were created differently compared to the first application (*Section* 6.4). In this second application, thanks to the relatively

[4] Ubuntu 14.04 32-bit, $^{\text{TM}}$ i5-2500 CPU @ 3.30GHz (only single core was used), 8GB RAM

[5] Note that it is not possible to render the whole graph with a significant level of detail on a single or double page.

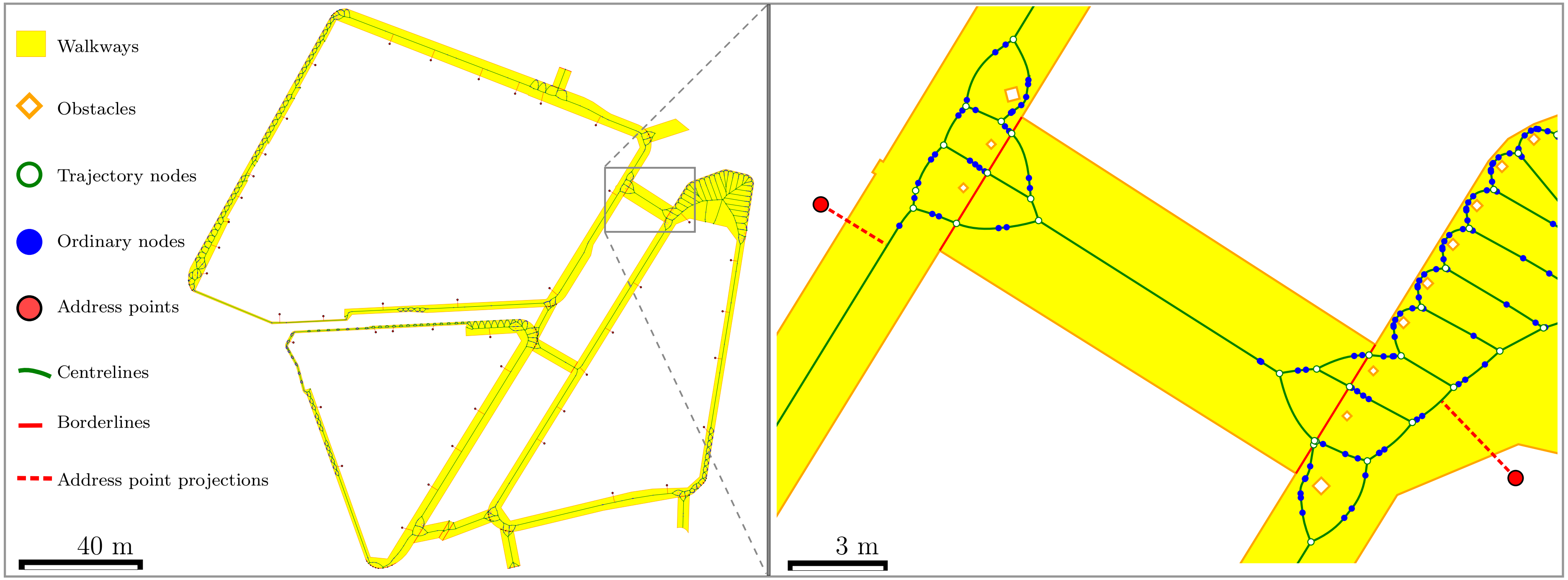


**Figure 6.12:** A part of the pedestrian network generated for the Saint Sulpice region in Paris. The figure on the left is the highlighted region in the *Figure* 6.8.

well-structured input vector data, we could apply the method described in *Section* 5.4.3. That is, semantic planar partitions were created automatically by directly importing the input vector data.

## 6.5.1 Creation of the Semantic Planar Partition

Four different vector datasets were used to create the semantic planar partitions. The details of the creation and preparation of these datasets are described in the following three subsections (6.5.1.1, 6.5.1.2, 6.5.1.3) but before some preliminary information is given in *Table* 6.3.

| ***Name*** | ***Type*** | ***Source*** |
|---|---|---|
| CASQY dataset | Polygonal | Provided by CASQY city hall |
| Pedestrian crossings | Polygonal | Manually created on laser orthophotos |
| Obstacles | Polygonal | Created using $\alpha$-shapes of a 2d point cloud |
| BD Adresse® | Punctual | IGN |

**Table 6.3:** Four vector datasets used for creating the CASQY semantic planar partitions.

The 2D arrangements supporting the planar partitions are composed of (11570 vertices, 11969 edges, 1112 faces) for region-1 and (61436 vertices, 63523 edges, 6978 faces) for region-2[6]. These arrangements were created in parallel to the semantic objects while they are imported from the input vector data. Three types of semantic objects were defined for three different data sources. Semantic objects imported from CASQY dataset are classified as *walkways* and the other two types were *pedestrian crossings* and *obstacles* correlated to their dataset names. For the first semantic planar partition (region-1), 951 semantic objects were imported (45 pedestrian crossings, 46 walkways, 79 address points, 781 obstacles) and for the second planar partition (region-2) 5727 semantic objects were imported (50 pedestrian crossings, 164 walkways, 460 address points, 5153 obstacles)[7].

*Figures* 6.13 and 6.14 display the automatically created semantic planar partitions using the four datasets.

### 6.5.1.1 Creating the Basic Semantic Planar Partition

The CASQY dataset describes the basic land use of the region and surprisingly the contained vector data are close to form a planar partition. Although the dataset was created with a non-topological GIS, there were not many errors[8] most of which are fixed by utilizing the PostGIS Topology[9] package using manual and automatic tools. In fact, these errors could have been fixed within StreetMaker but since we wanted to follow a different

[6] Note that these numbers do not satisfy the Euler's formula (*Equation* 1) since corresponding planar partitions are not composed of a single connected planar graph, instead they are a collection of connected planar graphs.

[7] Not all of the imported obstacles were on the walkways. The ones that were outside the walkways were discarded from the pedestrian network generation process but they are included in the total number of imported obstacles.

[8] There were a couple of hundreds of overlapping polygons and some gaps between the polygons.

[9] http://postgis.net/docs/manual-dev/Topology.html

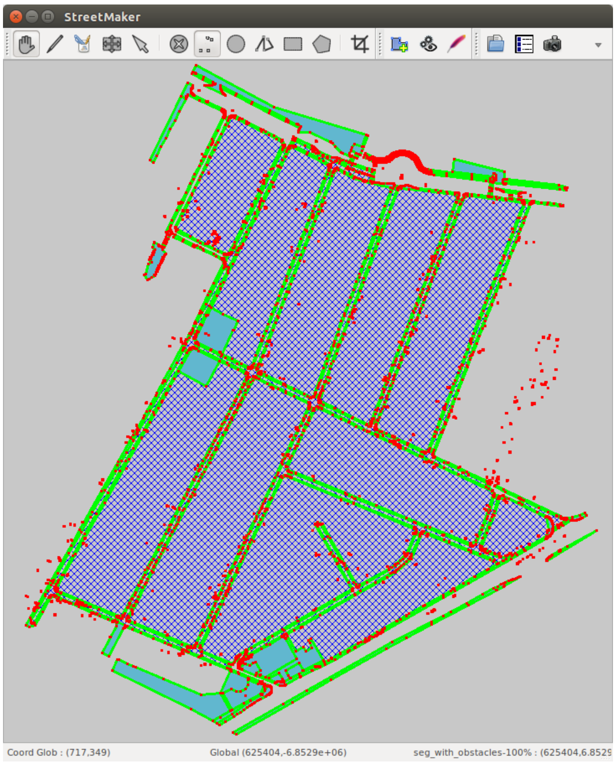

StreetMaker
Coord Glob : (717,349)
Global (625404,-6.8529e+06)
seg_with_obstacles-100% : (625404,6.852

**Figure 6.13:** Semantic planar partition for region-1 in CASQY: four datasets were used to create the semantic partition. Although there exist three types of semantic objects, only the walkways are clearly visible in this figure and in *Figure* 6.14 due to the relatively small scales of the pedestrian crossings and obstacles.

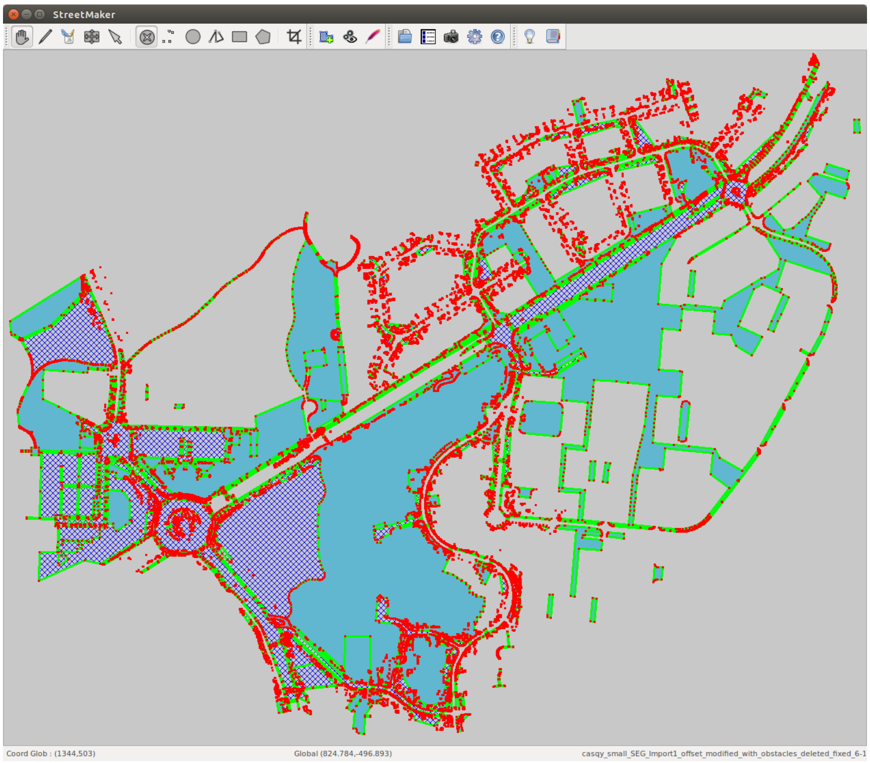

StreetMaker
Coord Glob : (1344,503)
Global (824.784,-496.893)
casqy_small_SEG_Import1_offset_modified_with_obstacles_deleted_fixed_6-1

**Figure 6.14:** Semantic planar partition for region-2 in CASQY: four datasets were used to create the semantic partition.

pipeline than the first application, external tools are used to pre-process the input vector datasets.

*Figure* 6.15 displays the two interested regions in the CASQY dataset. In this dataset, each polygonal feature has a type attribute. Semantic objects were imported from the dataset based on a type-filter. *Table* 6.4 displays the different types of geometric features and indicates the selected types that were imported into semantic planar partitions as semantic objects of type *walkways*.

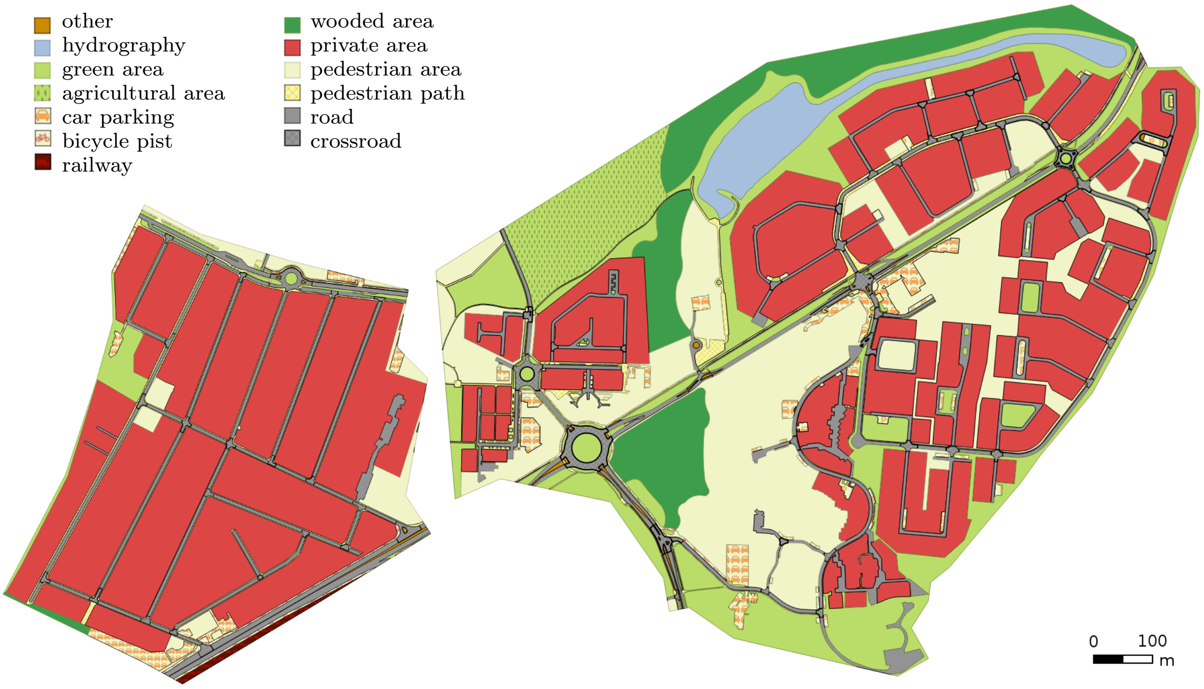


**Figure 6.15:** Two regions of CASQY dataset that were imported into StreetMaker for creating the semantic planar partitions displayed in *Figures* 6.13 and 6.14.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| crossroad | × | car parking | ✓ | pedestrian path | ✓ | other | × |
| bicycle pist | ✓ | green area | × | wooded area | × | railway | × |
| road | × | pedestrian area | ✓ | private area | × | hydrography | × |
| agricultural area | × | | | | | | |

**Table 6.4:** CASQY dataset type attribute: Each polygonal feature in the CASQY dataset has a type attribute. Different values of this attribute are given in the table. Polygonal features imported as walkways are marked with ✓and the rest of them are marked with ×.

#### 6.5.1.2 Importing Pedestrian Crossings

Almost all of the walkwable surfaces were imported as *walkways* from the CASQY dataset, however the CASQY dataset lacks the pedestrian crossings without which the pedestrian networks would not be much useful. Besides, there were not any previously created vector data containing the pedestrian crossings. Therefore, they had to be created manually.

IGN has a street-based Mobile Mapping System (MMS) called STEREOPOLIS [Paparoditis et al., 2012] which can capture image and lidar data. STEREOPOLIS collected spatial data from the CASQY region from which laser orthophotos were generated using the methods defined in Brédif et al. [2015] and Vallet and Papelard [2015]. Then, pedestrian crossings were captured manually (using QGIS[10]) with the guidance of the laser orthophotos (*Figures* 6.16 and 6.17).

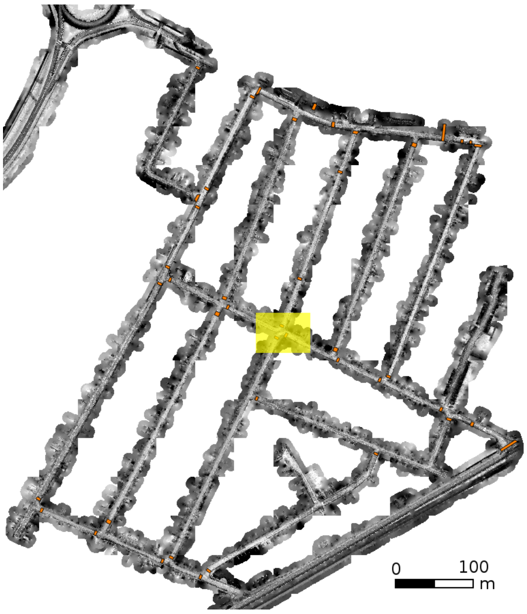


**Figure 6.16:** Laser orthophoto and manually drawn pedestrian crossings for CASQY region-1. The highlighted region is displayed in *Figure* 6.17 for a closer look.

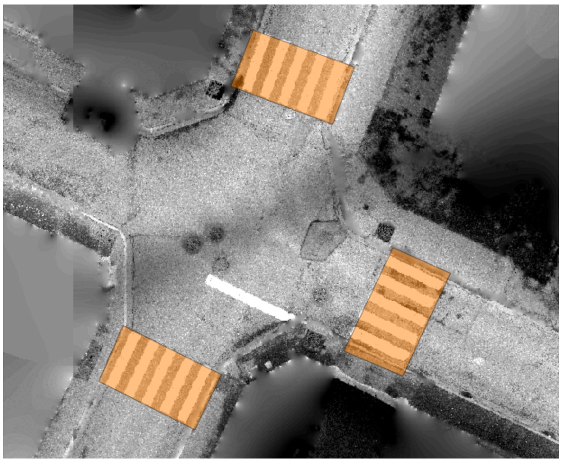


**Figure 6.17:** A closer look into the highlighted region in *Figure* 6.16.

Captured pedestrian crossings were imported into the semantic planar partitions that were containing only the walkways after the first mini-pipeline described in the previous *Section* (6.5.1.1). Many of the disconnected walkways were connected with the newly constructed pedestrian crossings.

[10] http://www.qgis.org/en/site/

#### 6.5.1.3 Creating and Importing Obstacles

The static obstacles on the walkways were also generated using the 3D lidar point cloud data collected by STEREOPOLIS. First, 3D points classified as static obstacles [Serna and Marcotegui, 2014] were projected vertically and the corresponding 2D silhouette points are obtained. Then, the $\alpha$-shapes algorithm (*Sections* 4.2 and 5.6.1) was utilized (with $\alpha$ equals to 20 cm) to estimate the shapes of the obstacles. Finally, estimated obstacles were imported on the walkways and the pedestrian crossings which completed the creation of semantic planar partitions. The overall view of the created semantic planar partitions are already given in the *Figures* 6.13 and 6.14. On the other hand, *Figures* 6.19 and 6.21 display much more closer views from the created semantic planar partitions and additionally, *Figures* 6.20 and 6.22 display the corresponding regions in the generated pedestrian network graphs.

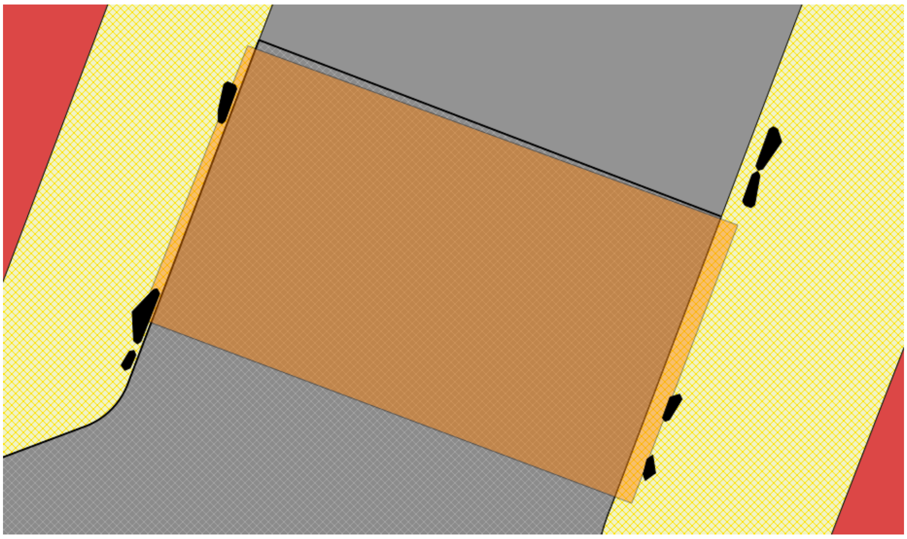


**Figure 6.18:** A close look at the three datasets used to create the semantic planar partitions. The polygonal datasets are displayed as layers on top of each other (in QGIS). The order from bottom to top is: CASQY dataset, pedestrian crossings (transparent brown) and obstacles (black). See also *Figures* 6.19 and 6.20 for the corresponding semantic planar partition and the generated pedestrian network graph.

The second pipeline explained in this section is more effective than the first pipeline discussed previously (6.4). Since the data at hand were more structured than the data in the first application, it took less time to create the semantic planar partitions. The total time for executing (using the same operating system and hardware as in the first application) the whole pipeline was around 5 hours including the preprocessing the initial CASQY dataset ($\sim$ 1 hour), manually drawing the pedestrian crossings ($\sim$ 1.5 hours), estimating the shapes of obstacles from the 2D point cloud ($\sim$ 1 hour), importing the vector data and creating the semantic objects ($\sim$ 30 min.), running the pedestrian network generation algorithm for CASQY region-1 ($\sim$ 5 minutes) and region-2 ($\sim$ 30 min.). Note that the run times are not directly related to the number of geometric primitives but also their configuration with respect to each other (e.g. the number of holes within a face, the number of reflex vertices in the polygons, etc.).

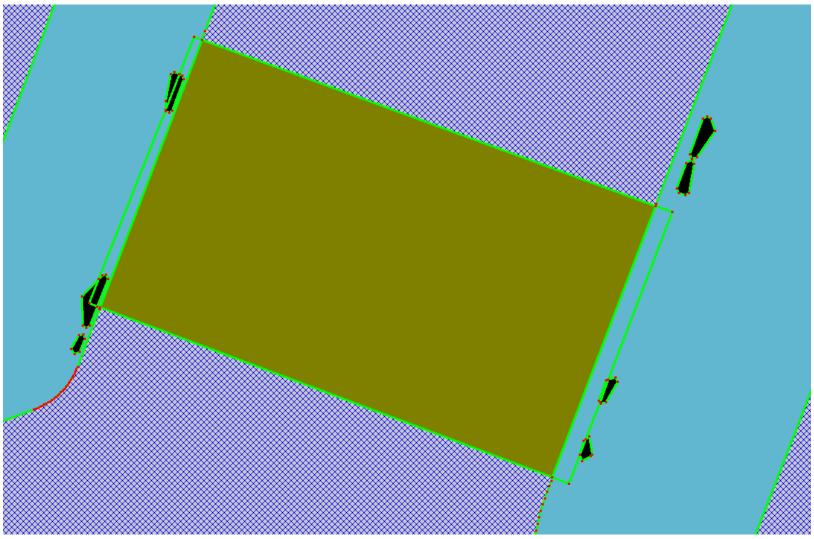


**Figure 6.19:** A close view of the CASQY semantic planar partition: walkways (cyan), pedestrian crossings (olive), obstacles (black). See also *Figures* 6.18 and 6.20 for the corresponding input datasets and the generated pedestrian network graph.

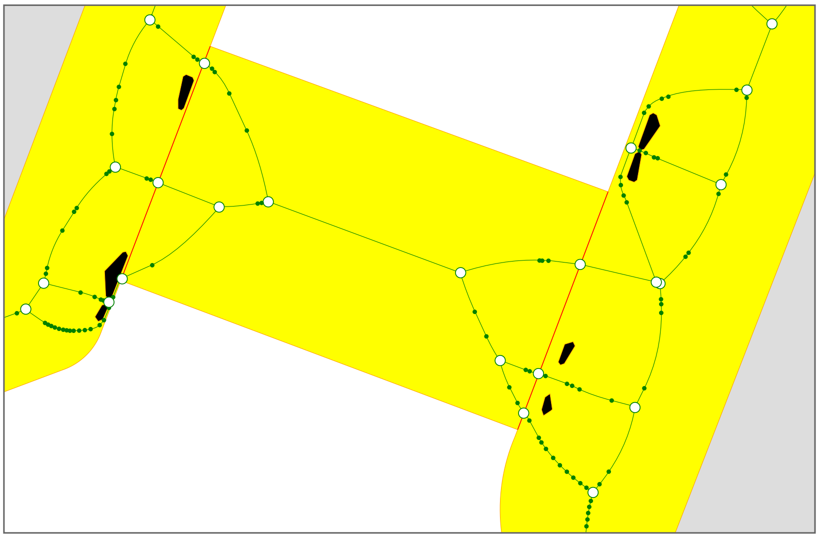


**Figure 6.20:** A close view of the pedestrian network graph computed from the CASQY semantic partitions: walkways including the pedestrian crossings (yellow), obstacles (black), borderlines (red), graph edges (green), graph nodes (green for ordinary nodes and white nodes stroked with green for trajectory nodes). See also *Figures* 6.18 and 6.19 for the corresponding input datasets and the semantic planar partition.

## 6.6 Implementation Details

### 6.6.1 Merging Delaunay Triangulations

STEREOPOLIS[Paparoditis et al., 2012] collects and stores lidar points in separate files. These files can be merged and processed at one time, however a better choice might be keeping them separate if one can process the separate files in parallel and merge the results. As mentioned before (*Section* 6.5.1.3), we wanted to estimate the shapes of the obstacles using the $\alpha$-shapes which depends on the Delaunay triangulation of the point

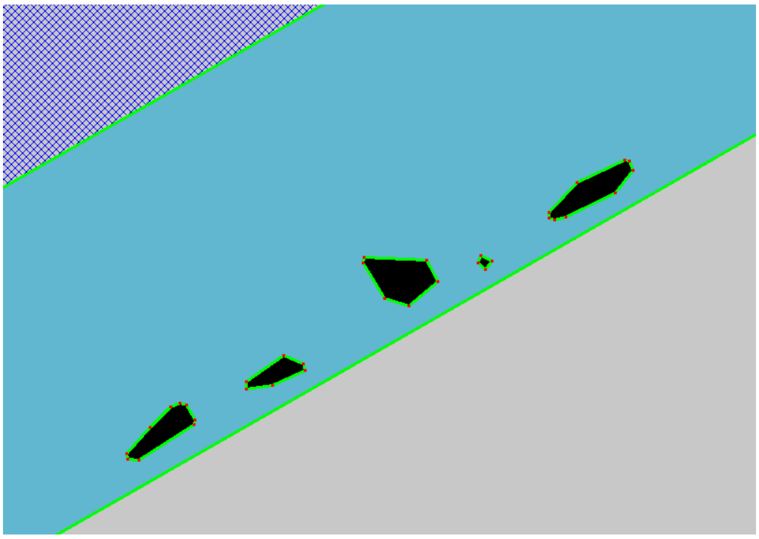


**Figure 6.21:** Another close view of the CASQY semantic planar partition: walkways (cyan), obstacles (black). See also *Figure* 6.22 for the corresponding generated pedestrian network graph.

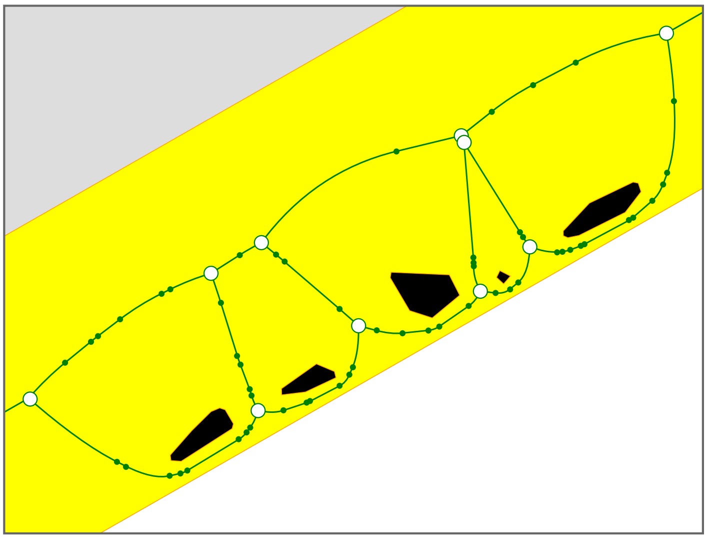


**Figure 6.22:** Another close view of the pedestrian network graph computed from the CASQY semantic partitions: walkways including the pedestrian crossings (yellow), obstacles (black), graph edges (green), graph nodes (green for ordinary nodes and white nodes stroked with green for trajectory nodes). See also *Figure* 6.21 for the corresponding semantic planar partition.

sets. Delaunay triangulation of separate files can be computed in parallel and then the resulting triangulations can be merged using the implemented algorithm developed by Lee and Schachter [1980]. The steps of this merging algorithm are described in detail along an example in *Appendix* B B.2. The parallel processing technique described here is one of the steps that we will need in scaling our pipeline. In our application however, we did not need parallel processing since the region that we worked on is relatively small.

### 6.6.2 Computing the Union of the Object Surfaces

Remember that at the lowest level a semantic object is defined by a set of faces in a 2D arrangement and the faces of an object are not necessarily connected. First, each connected group of faces belonging to the same object are recursively detected using the topological relations embedded in the halfedge data structure of the 2D arrangements. Then, these faces are converted into polygon with holes after applying an antenna removal procedure which removes possible antenna like structures those invalidating the polygon with holes. This first set of polygon with holes are kept in memory since they are also used to calculate the boundaries between different objects. These polygon with holes representing each object are then merged into another set of polygon with holes representing the union of all the walk-able surfaces on which the medial axis (*Section* 5.6.2) and/or straight skeleton (*Section* 5.6.3) algorithms run.

### 6.6.3 Representing Parabola Segments with Quadratic Bézier Curves

The intersection points between the medial axis and the borderlines are computed using the Bézier arrangement package of CGAL. First, each line and parabola segment defining the medial axis is inserted into a Bézier arrangement. But before line segments are converted to linear Bézier curves and parabola segments are converted to quadratic Bézier curves (see *Figure* 6.23 for the illustration of the proof).

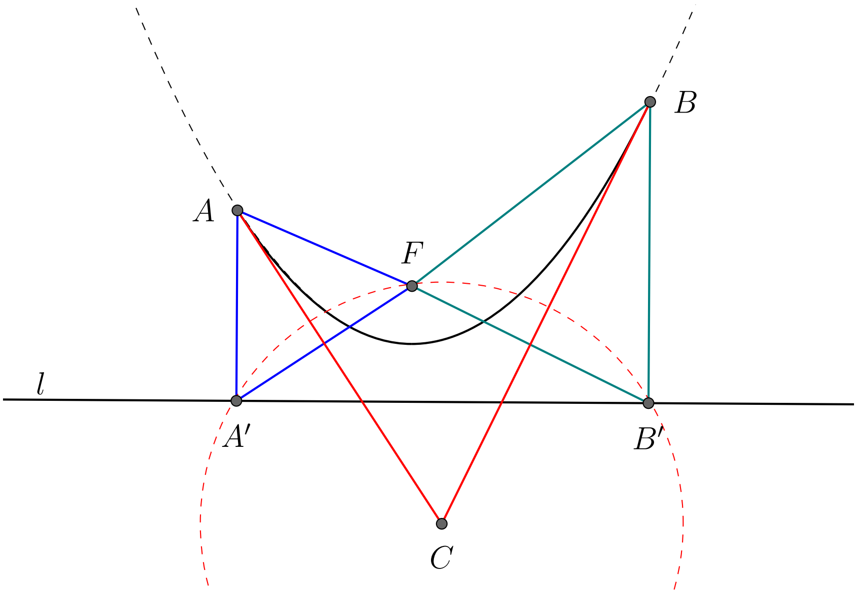


**Figure 6.23:** Representing a parabola segment with a quadratic Bézier curve: Let $\widehat{AB}$ be a parabola segment on a parabola defined by a directrix $l$ and a focus $F$. Also assume that $A'$ and $B'$ are the feet of the perpendiculars from $A$ and $B$ to $l$. By definition of the parabola, $\overline{AA'} = \overline{AF}$ and $\overline{BB'} = \overline{BF}$. Observe that the perpendicular bisectors of $\overline{A'F}$ and $\overline{B'F}$ meet at point $C$ which is also the center of the circle passing through $A', B', F$ (Observe that $\overline{A'C} = \overline{FC} = \overline{B'C}$). Notice that the bisectors are tangent to the parabola at $A$ and $B$ (by tangent bisection theorem Byer et al. [2010]). Hence, a quadratic Bézier curve can be defined with the control points $A, C, B$.

### 6.6.4 Length of Quadratic Bézier curves

The length of a parametric curve $(x = f_x(t), y = f_y(t))$ is given by the *Equation* 1 [Vince, 2013]. Let $B(t)$ be a quadratic Bézier curve with control points $P_0, P_1, P_2 \in R^2$. Then, the equation of $B(t)$ and its derivative are given by 2 and 3. Furthermore, let

$Q_1 = (q_{1x}, q_{1y}) = 2(P_0 - 2P_1 + P_2)$ and $Q_2 = (q_{2x}, q_{2y}) = 2(P_1 - P_0)$ be two points in $R^2$. Then observe that *Equation* 4 holds for $A = q_{1x}^2 + q_{1y}^2$, $B = 2(q_{1x}q_{2x} + q_{1y}q_{2y})$, and $C = q_{2x}^2 + q_{2y}^2$. Finally, the length of a quadratic Bézier curve computed using the formula 5.

$$s = \int_0^1 \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2} dt \tag{1}$$

$$B(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2 \tag{2}$$

$$\frac{\partial B(t)}{dt} = 2t\left(P_0 - 2P_1 + P_2\right) + 2P_1 - 2P_0 \tag{3}$$

$$\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2} = \sqrt{At^2 + Bt + C} \tag{4}$$

$$\begin{aligned} s &= \int_{t=0}^{t=1} \sqrt{At^2 + Bt + C} dt \\ &= \left(\frac{4AC - B^2}{8A^{3/2}}\right) \ln\left|2At + B + 2\sqrt{A(At^2 + Bt + C)}\right| \quad \Big|_{t=0}^{t=1} \end{aligned} \tag{5}$$

### 6.6.5 Projecting Points on Quadratic Bézier Curves

Let $Q$ be an arbitrary point in $R^2$ and $B(t)$ be a quadratic Bézier curve with control points $P_0, P_1, P_2 \in R^2$. The projection of $Q$ on $B(t)$ can be computed by solving the following optimization problem 6. This formulation leads to a cubic polynomial displayed in 7 where $a = P_0 - 2P_1 + P_2$, $b = P_1 - P_0$, and $c = P_0 - Q$. The projection point can be found by analysing the roots of this polynomial.

$$\underset{t}{\text{minimize}} \quad \|B(t) - Q\|^2 \quad \text{subject to} \quad t \in [0, 1] \tag{6}$$

$$a^2t^3 + 3abt^2 + (2b^2 + ac)t + bc = 0 \tag{7}$$

### 6.6.6 Splitting Quadratic Bézier Curves

CGAL Bézier arrangement package requires the inserted curves to be x-monotone. If this is not the case, then it automatically splits the inserted curves at the vertical tangent points. Moreover, each newly inserted curve may split the already existing ones. However the utilized library only calculates the split points but not the newly generated control points. These control points are required when exporting the computed pedestrian network graphs which are found using the well known De Casteljau's algorithm.

## 6.7 Conclusion

In this chapter, we introduced our proposed pedestrian network generation process and demonstrated how it is applied for two real life projects. In addition, we also presented two different pipelines for creating the required semantic planar partitions for pedestrian network generation.

The created graphs are the first examples of geometrically correct pedestrian networks. That is, the graph nodes are geo-referenced and the graph edges represent the true centrelines of the walkways. In addition, the geometric computations are done using the exact computation paradigm. The resulting graphs are further processed to identify homogeneous regions called trajectory arcs such that a travelling entity can move freely on a trajectory arc without giving any further navigational decisions. Furthermore, for each trajectory arc, the total length and the minimum distance to the surrounding object borders are also computed. These properties might be very useful for autonomous / semi-autonomous devices such as robots, smart electronic wheelchairs and personal transportation vehicles (e.g. two-wheeled self balancing devices) that use the generated graphs for navigation.

We have seen (*Section* 6.1) that in the literature there exist two types of pedestrian network graphs with different levels of detail. The detailed network graphs are generally designed for visually or mobility impaired pedestrians along with other types of helping devices. The graphs used for these purposes were created manually using GISs. In addition, we also quoted from some researchers who expressed the needs of new methods and tools for generating detailed pedestrian network graphs. Our work is the first effort in automatically generating such detailed graphs.

The proposed pedestrian network generation process depends on the input semantic planar partitions which are mainly composed of areal semantic objects. The semantic and thematic information captured by these objects are used to enrich the graph contents. For instance, there exist graph nodes for every intersection point between the walkway centrelines and the object borderlines. Therefore, such a node indicates a change from one object to another. In addition, any semantic information that can possibly supply particular knowledge/intelligence is embedded into the graphs. Consequently, if the initial partitions are maintained and the semantic information is gathered in time, the corresponding pedestrian networks can be recomputed to utilize the gathered semantics.

The object resolution of the input semantic planar partitions define the resolution of the generated graphs. That is, the more detailed input partitions, the more correct and enriched graphs we get. Having finer details in the semantic planar partitions do not harm if a less detailed graph is wanted. Semantic objects used to compute the graphs can be filtered according to their types. For instance, if a static obstacle avoiding graph is desired, the barrier objects preventing the movement of the pedestrians can be excluded from the walkable surfaces, therefore the surfaces occupied by them will not be included in the centreline calculation. On the other hand, these objects can be included in the process (by not excluding) if we do not need such detailed graphs.

Although we applied the pedestrian network generation process for relatively large regions, as the region of interest gets larger, our process will be limited due to the required computation time. For processing larger regions such as the whole cities, some sort of parallel programming (single machine multiple cores, distributed computing, grid computing and cloud computing) is required. Another limitation is the stability of the medial axis algorithm used to compute the centrelines of the walkways. That is, very small features (e.g. a tiny spike) at the borders of the objects cause useless branches in the generated graphs. Smart ways of eliminating these unwanted branches can be searched as a future work. The first trial to solve this problem might be finding the graph nodes with degree greater than two and then checking the lengths of the connected edges until a degree one node is reached. The branches whose total lengths are below a certain threshold can be removed from the graphs.

In our applications, we did not encounter any pedestrian bridges going on top of other pedestrian paths. Such a case would cause a problem in our current design, since it is limited to 2D. In order to handle such cases, we need at least 2.5D arrangements. A 2.5D arrangement can be achieved by embedding the height information into the arrangement vertices.

In both applications we projected the positions of building doors onto the generated graphs to find the nodes from which the pedestrians can enter into the network. Similarly, we could have also projected the locations of bus stops, metro and tramway stations to find out the connection points between different types of networks. These connections will be very useful for full, city level navigation maps so that many alternative routes can be calculated with different options. In addition, indoor maps can also be used as a part of bigger semantic planar partitions. For instance, a train station, a commercial center or administrative building plans can be added to the partitions as separate objects. In this case, we need to differentiate two different types of borders between two objects (e.g. a sidewalk and a building): borders that are allowed (the doors of the building) and not allowed (the walls of the building) to be crossed by a travelling entity. Moreover, remember that additional rules can be applied to the boundary crossings with the connectivity graphs.

Similar pedestrian network graphs can also be computed for indoor environments such as big governmental buildings, museums, commercial centres, etc. These graphs together with indoor positioning systems (e.g. a portable device carried by a pedestrian locating its position with respect to an indoor map and a network graph) will be very useful both for people and autonomous devices such as cleaning robots and carrier robots. Finally, pedestrian graphs can be used in pedestrian simulation applications and computer games. For instance, navigation meshes are generally used in computer games (to define which areas of an environment are traversable by agents) can be replaced with a semantic planar partition and a corresponding network graph.

# Chapter 7

# 3D Generalized Cylinder Modelling from a Single Image

## 7.1 Introduction

In the previous chapters, we have discussed 2D GIS application development process along with the static obstacle avoiding pedestrian network generation. We have seen that the models of the 2D ground surfaces are needed in order to compute the desired network graphs. Similarly, for 3D GIS applications, the environments should be modelled in 3D. A 3D environment is composed of a 2D ground surface embedded in 3D and the 3D objects are positioned on the ground surface. There exist many different types of objects (e.g. buildings, trees, city furnitures, traffic lamps and signs, etc.) which can be represented / modelled best (relatively) with different techniques[1]. In this chapter, we will present the preliminary work and research that has been carried out in order to develop a 3D modelling system for 3D GIS applications.

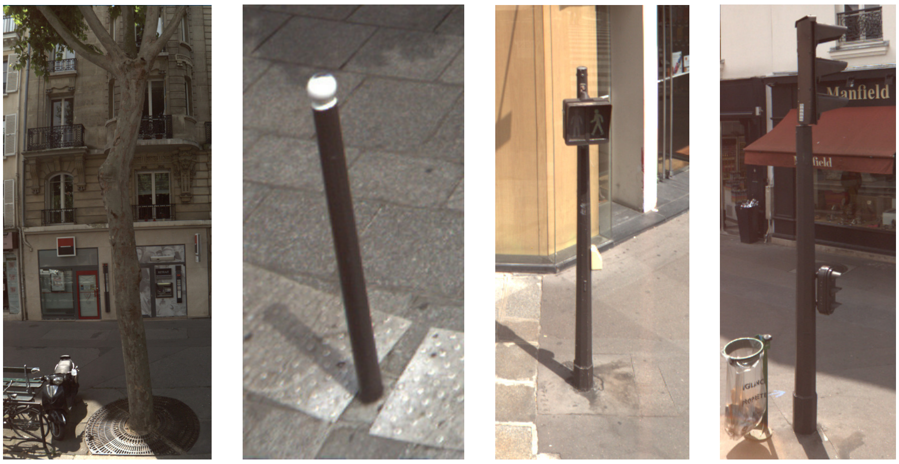
There exist two fundamental types of data that are commonly used in 3D modelling and reconstruction. These data are images and laser point clouds. In the literature, numerous works using either or both of these data have been published: modelling from single images [Oh et al., 2001; Lau et al., 2010; Xue et al., 2012; Chen et al., 2013], modelling from multiple images [Debevec et al., 1996; Pollefeys and Gool, 2002; Seitz et al., 2006], modelling from laser point clouds [Fangi et al., 2001; Li et al., 2011] and modelling from both [Syed et al., 2005; Pylvanainen et al., 2012]. Our final goal is the development of a semi-automatic system which enables easy 3D modelling from both of these data sources. Such a system can be integrated into a web-based 3D application (e.g. Google street view [Anguelov et al., 2010] and IGN's itowns [Nguyen et al., 2015]) so that the users can collaborate to generate photo-realistic virtual city models. Reaching this ultimate goal is a challenging task and it is difficult to achieve as an additional part of a PhD work. However, the task at hand is composed of many interesting research topics and we chose

[1] Stroud [2006] classified representations for solid modelling into four main groups: cell decomposition, general sweeping, set theoretic (Constructive Solid Geometry (CSG) and half-space modelling technique) and boundary representation. Besides these, surface mesh models are common especially in computer graphics. Interested reader may also refer to Mäntylä [1988], Mortenson [1997] and Campbell and Flynn [2001] for more information and broad surveys on the 3D model representation in geometric modelling and computer graphics.

to work on the 3D generalized cylinder modelling from single view images as a starting point to such an advanced modelling system.

From a single image, an object model can only be reconstructed upto a scale factor if there is not any prior information about the size of the object or its distance from the camera. For a correctly scaled geo-referenced model, at least two-views of the object and the camera parameters are needed. Once the shape of the object is reconstructed upto a scale factor, geo-referencing and scaling can be done afterwards which is left as a future work. Notice also that, integrating other views of the object into the modelling pipeline should improve the geometric quality of the reconstructed model. In addition, some of the dimensions of the object cannot be reconstructed from a single-view when the object's geometry lacks symmetry.

Objects of type lampposts, poles, tree trunks, signposts, etc. are all in the form of a generalized cylinder (*Figure* 7.1) and these are among the most frequent objects that we encounter on the surface of the walkways. Recall that for the pedestrian network generation application, we need to construct the outlines of the obstacles on the walkways. In fact, once the modelled objects are geo-referenced and scaled accordingly, the footprints of these objects (a 3D circle on the ground surface for this case) can be utilized to define obstacles in the pedestrian network generation application.



(a) tree trunk (b) pole (c) signpost (d) lamppost

**Figure 7.1:** Examples of geospatial objects in the form of generalized cylinder.

Kolbe et al. [2005] defined five levels of detail for 3D virtual city models ranging from 0 to 4. Level-0 is the coarsest level describing only the terrain (ground surface) over which an areal image or a map can be draped. Level-1 city models possess block building models without any texture or roof structure whereas level-2 city models have different roof structures and textures on the buildings. Level-3 increases the details on the building walls (balconies, bays, projections) and roof structures on which high resolution textures can be applied. Additionally, detailed vegetation and transportation objects are also modelled. And at the highest level (level-4), interior structures to the buildings (e.g. stairs, rooms, doors, furnitures) are added. Although, it is not explicitly mentioned, one can consider the street level small-scale objects as a part of level-4 since their in-door counterparts exist in the same level. Therefore, our attempts described in this chapter can be considered within this context, i.e. level-4 virtual city modelling. Interested reader may refer to Thomas and Donikian [2000], Döllner et al. [2006], Ross [2010] and Singh et al. [2013] for generating the 3D city models at various levels of detail.

This chapter continues with *Section* 7.2 in which our modelling system is described. In *Section* 7.2.1, first a generative definition of generalized cylinders is given, then a brief state of the art is presented. The main focus of this section is on a recent paper [Chen et al., 2013] that our preliminary work can be compared with. The next section (*Section* 7.2.2) explains the representation of generalized cylinders in our system. Then the first step of the generalized cylinder modelling is illustrated in *Section* 7.2.3. In this first step, the user is asked to draw a 2D ellipse on the image which is the projection of a 3D circle. In the following section (*Section* 7.2.4), the analytic methods of computing such 3D circles from their perspective and orthographic projections are discussed in detail. The computed 3D circle is used as the first 3D cross section in the generalized cylinder that is to be modelled. Then the user sweeps the major axis of the initial elliptic 2D profile along the image of the generalized cylinder while the dragged major axis is adjusted to fit the silhouette of the generalized cylinder. The data generated during this sweep is used to estimate the final 3D model of the generalized cylinder together with a user selected prior constraint on the axis of the generalized cylinder. This final step is discussed in *Section* 7.2.5 and the chapter is concluded (*Section* 7.4) with the discussion of contributions and perspectives.

## 7.2 3D Generalized Cylinder Modelling from a Single Image

### 7.2.1 A Brief State of the Art

Binford [1971] inspired by the medial axis transform [Blum, 1967] and introduced the term "generalized cylinder". The idea was originated from sweeping a planar curve (cross-section) along an axis or spine which generates an object of type generalized cylinder. That is, a generalized cylinder is defined by a space curve called the axis of the object, and a set of cross-sections. According to the properties of the cross-sections and the axis of the objects, Shafer [1985]; Naeve and Eklundh [1995] presented detailed generalized cylinder taxonomies. The first group of generalized cylinders can be classified according to the constraints defined on the axis. The axis can be a straight line (Straight Generalized Cylinder (SGC)), planar curve, general space curve, open curve or closed curve (Toroidal Generalized Cylinder (TGC)). Similar constraints can also be defined on the cross-sections. For instance, cross-section planes can be perpendicular to the tangents to the axis (Right Generalized Cylinder (RGC) - opposite: Oblique Generalized Cylinder (OGC)). Moreover, cross-section shape types can also be used to further classify the generalized cylinders: cross-sections can only be circles (Circular Generalized Cylinder (CGC)), polygons (Polygonal Generalized Cylinder (PGC)), open (cross-sections are not Jordan curves - not simple closed curves) and closed. Finally, some subclasses of generalized cylinders can be formed by restricting the transformation from one cross-section to the another. For instance, a Homogeneous Generalized Cylinder (HGC) allows only uniform scaling of the cross-sections, i.e. all cross-sections have the same shape but vary in size. On the other hand, a Uniform Generalized Cylinder (UGC) enforces all the cross-sections to be identical in size and shape. Interested reader may refer to aforementioned references for the details and the full taxonomy charts. In the context of this chapter, we are interested in Circular Homogeneous Generalized Cylinders (CHGCs) and from now on unless otherwise is stated, the term “generalized cylinder” should be accepted as CHGC.

Modelling objects from single-view images is an ill-posed problem due to the lost of depth information during image formation (perspective transformation). Therefore, all of the works that have been done in this field came with various assumptions and constraints to infer the 3D geometry of a scene or an object. Interested reader may refer to Oswald et al. [2013] for a survey of the state of the art single-view modelling methods and algorithms. In this chapter, our focus is on the generalized cylinder modelling from single-view images.

A sub-class of generalized cylinders called Straight Homogeneous Generalized Cylinders (SHGCs) (esp. a subgroup of SHGCs called solids of revolution) have been studied a lot. Ponce et al. [1989] analysed the orthographic projections of SHGCs (with the assumption: cross-sections are perpendicular to the axis, i.e. right SHGCs) using differential geometry and identified several invariants which can further be extended to perspective projection. In deed, Richetin et al. [1991] extended the results of Ponce et al. [1989] for perspective camera models. Later, other researchers developed methods for modelling SHGCs from single-view images [Sato and Binford, 1993; Ulupinar and Nevatia, 1995; Zerroug and Nevatia, 1999] that exploited these works. Some other notable works on the topic can be listed as Gross and Boult [1990, 1996]; Sayd et al. [1996]; Utcke and Zisserman [2003]; Colombo et al. [2005].

Chen et al. [2013] developed an interactive part-based system for modelling 3D man-made objects (objects that are composed of primitive parts: generalized cylinders, cuboids and spheres) from a single image. First, object parts are modelled separately and then geo-semantic constraints (parallelism, orthogonality, collinear axis endpoints, overlapping axis endpoints, coplanar axis endpoints and coplanar axes) are introduced between the object parts based on some anchor points defined on the parts. Users can manually edit these automatically inferred constraints which are later used in an optimization process[2] to solve the position and orientation of the modelled parts. Furthermore, a local coordinate frame is defined (*Figure* 7.2(a)) for each part which is used to parametrize[3] the anchor points with respect to the depths of the local coordinate frame centres. These parametrizations are used to keep the projections of the parts consistent with the image during the optimization process which tries to localize 3D parts that satisfy the geo-semantic constraints.

The previous paragraph summarizes the over-all process applied by Chen et al. [2013] for modelling a 3D object which includes the modelling of individual parts separately and then merging the parts to compose the final object based on an optimization problem. In this paragraph, the modelling of the primitives of type generalized cylinders are explained from which we have inspired to design our modelling system. Users initiate the modelling by drawing an elliptic 2D profile on the input image. This 2D profile corresponds to the perspective projection of the first 3D circle on the generalized cylinder (*Figure* 7.2(a)). This initial 3D circle is estimated such that its centre is located on the viewing plane (image plane) and its normal and radius are assigned according to the length and orientation of the major and minor axes of the 2D profile. The user then completes the generalized cylinder modelling by sweeping the 2D profile along the image of the gener-

[2] Some internal constraints for each individual part are also added to this optimization which are mentioned in the corresponding article but not given explicitly.

[3] These parametrizations are obtained using the orthogonality constraints between the frame axes and the inverse perspective projection equations.

alized cylinder. This sweep generates an approximation to the axis of the 3D generalized cylinder under the constraint of planar axis and the axis plane is parallel to the viewing plane. The generated axis is sampled uniformly (at every 5-pixel) in the image space and at each point a copy of the previously estimated 3D circle is centred. The normals of the copied 3D circles are aligned according to the bending / orientation of the axis curve at that point and their radii are adjusted to meet the generalized cylinder's outline in the image (*Figures* 7.2(b) and (c)). As a result, a set of 3D circles are generated along the approximated axis of the modelled generalized cylinder. The positions of these 3D circles are later updated within the aforementioned optimization process after which the 3D circles can be warped to 3D ellipses. The methods utilized by Chen et al. [2013] will be further explained and compared to our methods along the *Sections* 7.2.4.3 and 7.2.5 once the foundations of the problem are discussed.

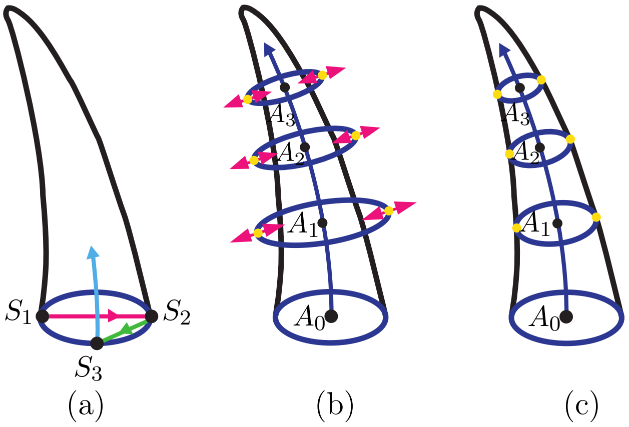


**Figure 7.2:** 3-sweep generalized cylinder modelling: first two lines ($S_1S_2$ and $S_2S_3$) and the sweep of the axis. $S_1S_2$ and $S_2S_3$ define the first 2D elliptic profile and the four points ($S_1$, $S_2$, $S_3$, the first sampled point along the sweep) define the perspective projection of the local coordinate frame of the generalized cylinder (a). At each sampled point on the axis, a copy of the previously created 3D circle is positioned (b). Then the copied circle is snapped to the generalized cylinder outline (c). This figure is taken from Chen et al. [2013] but the caption of the figure is re-written.

The discussed modelling system needs to identify the outlines of the imaged objects. Chen et al. [2013] first utilized a hierarchical edge feature extraction method from Arbelaez et al. [2011] and then merged the detected edges into continuous curves via Cheng [2009]. Therefore, a two-step preprocessing is required before starting to the modelling.

### 7.2.2 Representing 3D Generalized Cylinders

The generative definition of the generalized cylinders given by Binford [1971] requires an infinite number of cross-sections (3D circles in our case). A practical solution for realizing the generalized cylinders in computers is to approximate the axis by a set of connected linear space curves and placing a cross-section at every point where these linear segments meet. An example of a generalized cylinder constructed in this way is displayed in *Figure* 7.3.

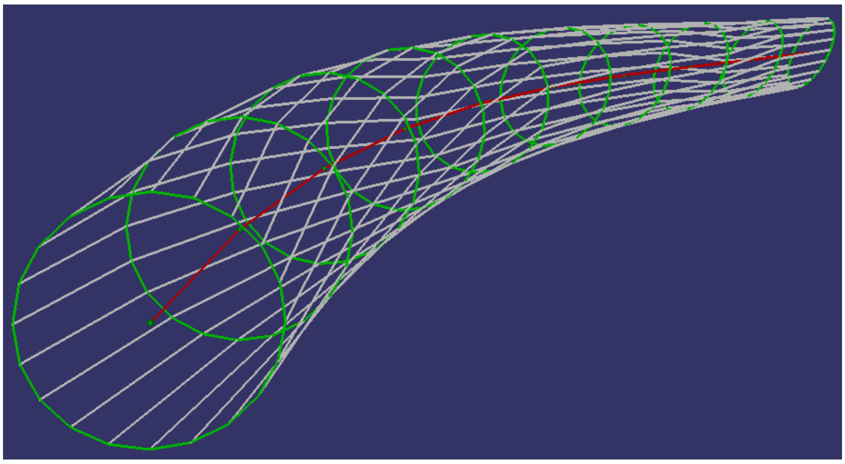


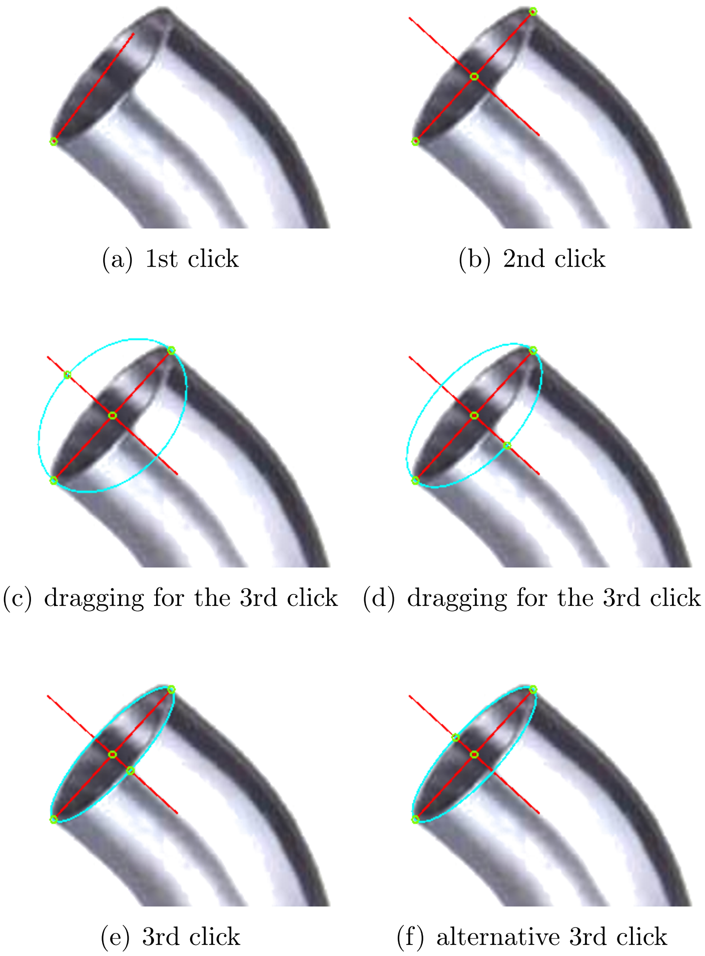
**Figure 7.3:** A homogeneous circular generalized cylinder: approximated by 10 (ten) cross-sections, therefore the axis of the generalized cylinder is defined by 9 (nine) 3D linear segments connected to each other. Colour codes: axis (red), cross-sections (green), curves on the surface of the generalized cylinder (light gray).

### 7.2.3 Drawing the Initial 2D Elliptic Profile

The procedure for modelling a generalized cylinder is initiated when the user begins to draw the projection of the initial cross-section of the generalized cylinder on the input image. Three mouse clicks are required to accomplish this task. The first click is accepted as the major axis starting point of the 2D ellipse. While the mouse pointer is being dragged, the current major axis line is drawn in real time to give visual guide to the user (*Figure* 7.4(a)). The second click is taken as the endpoint of the major axis, which also defines the centre of the 2D elliptic profile (the midpoint of the first and second clicks). Furthermore, a minor axis guideline is also displayed to the user which is perpendicular to the major axis and its length is equal to the major axis (*Figure* 7.4(b)). After the second click, the user drags the mouse pointer over the image for the third click which finalizes the 2D profile drawing. During this dragging, the mouse pointer is projected onto the minor axis guideline and based on these projected points, candidate ellipses are dynamically displayed to the user (*Figures* 7.4(c) and 7.4(d)). For an ellipse, the length of the minor axis can be at most equal to the length of the major axis (i.e. the ellipse is a circle). Therefore, if the mouse pointer projection falls outside the minor axis guideline, it is assigned to the closest endpoint of the minor axis guideline.

Observe that the third click can be performed at both ends of the minor axis (i.e. as either in *Figure* 7.4(e) or *Figure* 7.4(f)). Although, the resulting ellipse on the image plane is same, we have different interpretations depending on which endpoint is selected by the user. This is related to the selection of the two possible 3D circles whose projections fit to the 2D ellipse. This will be further discussed in the *Section* 7.2.4.3 after analysing the estimation of 3D circles from their perspective projection (*Section* 7.2.4.1).

Once the 2D ellipse is drawn on the image, its algebraic equation ($ax^2 + bxy + cy^2 + dx + ey + f = 0$) can be computed from the ellipse parameters: centre ($\rho(\rho_x, \rho_y)$), semi-major axis length ($m$), semi-minor axis length ($n$) and the angle of rotation ($\theta$: the counter clockwise angle between the major axis and the positive x-axis). By starting from an ellipse in the standard form (i.e. the ellipse is centred at the origin and its axes are parallel to the coordinate frame axes) and applying the required transformations, one can



(a) 1st click

(b) 2nd click

(c) dragging for the 3rd click

(d) dragging for the 3rd click

(e) 3rd click

(f) alternative 3rd click

**Figure 7.4:** Drawing the initial 2D elliptic profile with three mouse clicks. A small green circle is positioned at the critical points: first click, second click, centre of the 2D ellipse, and the projections of the mouse pointer and the third click onto the minor axis guideline. Major axis and the minor axis guideline are displayed in red. The candidate and final ellipses are displayed in cyan.

compute the relations given in the *Equation* 1. In the next section (*Section* 7.2.4), we will see that the algebraic equation of the ellipse is needed for computing the 3D circles whose perspective projections are the ellipse.

$$\begin{array}{rcl} a & = & 0.5(m^2 + n^2 + (n^2 - m^2)\cos(2\theta)) \\ b & = & (n^2 - m^2)\sin(2\theta)) \\ c & = & 0.5(m^2 + n^2 - (n^2 - m^2)\cos(2\theta)) \\ d & = & -2a\rho_x - b\rho_y \\ e & = & -2c\rho_y - b\rho_x \\ f & = & a\rho_x^2 + b\rho_x\rho_y + c\rho_y^2 - m^2n^2 \end{array} \qquad (1)$$

### 7.2.4 Estimating a 3D Circle from Its Projection

In this section, we will analyse the estimation of 3D circles from their single view orthographic and perspective projections. In orthographic projection, 3D object coordinates are transformed on the viewing plane along parallel lines which intersect with the viewing plane at 90 degrees. On the other hand, in perspective projection, 3D object coordinates are transformed to the viewing plane along projection lines that converge at the centre of projection. We can say that the centre of projection for the orthographic projection is at the infinity.

Observe that a 3D circle can be moved along the viewing direction as the projection lines keep passing through the circle. This can happen if only the 3D circle is scaled during its movement. In addition, since the projection lines are passing through the circle, its projection on the image plane does not change. Therefore, we can conclude that there exist infinitely many 3D circles whose projection (valid for both perspective and orthographic projections) are same. On the other hand, we will see that for the general case (i.e. when the projection of the circle is an ellipse), when the depth (i.e. the component of the circle centre along the viewing direction) or radius of the circles are fixed, there exist two separate 3D circles whose projections are same.

#### 7.2.4.1 Estimating 3D Circles Under Perspective Projection

The perspective projection of a 3D circle on the image plane is an ellipse except for the two special cases. The first special case occurs when the plane of the 3D circle is parallel to the viewing direction, i.e. the circle normal is perpendicular to the viewing direction or the circle plane passes through the centre of the projection. In this first special case, the 3D circle is projected as a line segment on the image plane. In the second special case, a 3D circle is projected as a circle on the image plane. The special cases will be further discussed in the following paragraphs.

The problem of computing 3D circles from their perspective projections can be classified as solved after Safaee-Rad et al. [1992] presented an analytical solution to the problem. Yet other works have been published on the topic which applies different methods. For instance, Philip [1997] proposed a method which analyses the eigenvalues of a quadratic form that is associated with the projected ellipse. Then, he inserts the parametric equation of the 3D circle into this quadratic form and obtain six equations which constrain the parameters of the 3D circle. Based on these constraints and a geometric intuition he managed to solve the problem approximately (refer to the article for the error analysis). Ferri et al. [1993] proposed another method based on decomposition of a quadratic form into two linear forms which define two planes that support the 3D circles. This decomposition requires an optimization step and only after the circle centres are determined by intersecting these planes with the 3D viewing cone which passes through the projected ellipse and the circles. In our tests, this method generally works well but it has robustness issues especially when the elliptic projections are relatively small. This is caused by the optimization process which returns either local minimums or false global minimums due to relatively noisy data.

We utilized the analytical method of Safaee-Rad et al. [1992] which solves the problem in two steps using the pinhole camera model. In the first step, the orientation problem is solved (i.e. the circle normals are found), then the circle centres are computed in the second step. In this section, the problem will be solved for unit circles from which other circles can be obtained at the desired depth or radius by scaling. The analysis presented in this section similar to Safaee-Rad et al. [1992] but it is more detailed, the symmetry in the solution set is explicitly discussed and additionally the degenerate case is also included in the analysis.

The input to the problem is a given ellipse ($\mathcal{E}$, *Equation* 2) on the image plane and the focal length ($|k|$) of the pinhole camera (see *Section* 7.3 for the other needed parameters for a practical application). The desired solution is a set of unit 3D circles (let $\mathcal{C}$ be of one these circles) in the camera coordinate frame whose projections match with the given ellipse on the image plane.

$$\mathcal{E} : ax^2 + bxy + cy^2 + dx + ey + f = 0 \tag{2}$$

Let the camera is positioned at the centre of the camera coordinate frame and it is looking down the negative $z$-axis[4]. Therefore, the image plane is located at $z = k$ ($k < 0$). Observe in *Figure* 7.5 that the projection lines passing through both the 3D circle and the projected ellipse form a cone called viewing cone whose apex (or vertex) is at the centre of projection. Based on this observation, we can reformulate the 3D circle estimation problem: given a cone ($\Phi$, *Equation* 3)[5] with its vertex at the origin of the camera coordinate frame and its intersection with the image plane ($z = k$) is a known ellipse ($\mathcal{E}$), then find the set of unit 3D circles (position and orientation) that are generated by intersecting $\Phi$ with a corresponding set of planes (let $\Pi$ be one of these planes).

$$\Phi : X^TQX = Ax^2 + By^2 + Cz^2 + 2Fyz + 2Gxz + 2Hxy = 0 \tag{3}$$

where

$$X = \begin{bmatrix} x & y & z \end{bmatrix}^T \text{ and } Q = \begin{bmatrix} A & H & G \\ H & B & F \\ G & F & C \end{bmatrix}$$

The relations between the coefficients of $\Phi$ and $\mathcal{E}$ can be computed by replacing $z = k$ into the equation of $\Phi$ (*Equation* 4). As a result, from the given $\mathcal{E}$, the viewing cone can be constructed.

$$\begin{array}{rclrclrcl} a & = & A & b & = & 2H & c & = & B \\ d & = & 2Gk & e & = & 2Fk & f & = & Ck^2 \end{array} \tag{4}$$

The cross-product terms (if any) in the cone equation can be eliminated by a change of variables which can be achieved by applying the *Principal Axes Theorem* [Anton and Rorres, 2005]. Observe that $Q$ is a symmetric matrix and the principal axes theorem states that every $n \times n$ symmetric matrix has real eigenvalues and there is an orthonormal basis of its eigenvectors. Therefore, eigen decomposition of $Q$ can be computed as $Q = PDP^T$ where $D$ is a diagonal matrix whose entries are eigenvalues of $Q$ and $P$ is an orthogonal matrix whose columns are eigenvectors of $Q$. Assume that the eigenvalues of $Q$ are $\lambda_1, \lambda_2, \lambda_3$ and the corresponding eigenvectors are $e_1, e_2, e_3$. Then, the change of variables

[4] This is the default camera orientation in OpenGL.

[5] The equation of a cone whose vertex is at the origin of a coordinate frame [Sharma, 2005].

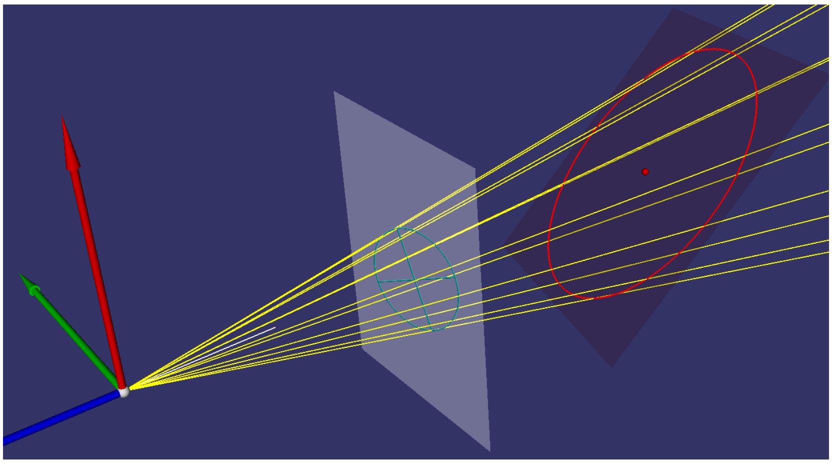


**Figure 7.5:** Perspective projection of a 3D circle: camera coordinate frame (XYZ → RGB), the projected ellipse (cyan) on the image plane, the 3D circle (red) and its supporting plane. Observe that projection lines (yellow) passing through both the 3D circle and the projected ellipse, constitute a cone (viewing cone) whose apex is at the centre of the camera coordinate frame.

(i.e. change of the coordinate system by rotating it) can be obtained by applying $X = PX'$ to the cone equation (*Equation* 5).

$$\begin{aligned} \Phi' : (PX')^T Q (PX') &= X'^T P^T Q P X' \\ &= X'^T D X' \\ &= \lambda_1 x'^2 + \lambda_2 y'^2 + \lambda_3 z'^2 = 0 \end{aligned} \tag{5}$$

where

$$X' = \begin{bmatrix} x' & y' & z' \end{bmatrix}^T, \; P = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} \text{ and } D = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

Observe that, in the new $x'y'z'$ coordinate frame, the cone ($\Phi'$) is free of cross-product terms (*Equation* 5). A cone equation free of cross-product terms is said to be in a standard form which has the following property: one of the three coefficients of the quadratic terms has a different sign than the other two. This property can also be derived directly from the *Equation* 5. If all the eigenvalues had the same sign, then the only eigenvalues that satisfy the equation would be all zero. Therefore, one of the eigenvalues should have a different sign than the others. As a result, the sign of the eigenvalues of $Q$ should be either $(\lambda_1, \lambda_2, \lambda_3) \rightarrow (+, +, -)$ or $(-, -, +)$. One can always multiply $Q$ by $-1$ in order to select one of the two sign configurations. Assume that the first configuration is selected.

Observe also that, six different $P$ and $D$ matrix pairs can be constructed by permuting the order of the eigenvectors and the corresponding eigenvalues. In half of these matrix pairs, $P$ is a rotation matrix ($det(P) = 1$) and it is a reflection matrix ($det(P) = -1$) for the remaining. We want $P$ to be a rotation matrix (for rotating the coordinate frame) since reflections reverse the clockwise / counter clockwise orientations. Among the remaining three possible configurations, we put $e_3$ into the third column of $P$ and select the configuration displayed in *Equation* 5. Note that, the axis of a cone in standard form is aligned with the coordinate axis whose coefficient has a different sign than the other two. According to our selection, the axis of $\Phi'$ is aligned with the $z'$-axis in the $x'y'z'$ coordinate frame.

The next step is to find a plane ($\Pi'$, *Equation* 6) in $x'y'z'$ coordinate frame such that it intersects with $\Phi'$ on a 3D circle. Also assume that $n' = \begin{pmatrix} a & b & c \end{pmatrix}$ is the unit normal of $\Pi'$.

$$\begin{aligned} &\Pi' : ax' + by' + bz' + d = 0 \\ &a^2 + b^2 + c^2 = 1 \end{aligned} \tag{6}$$

Intersecting $\Phi'$ and $\Pi'$ is not easy in the $x'y'z'$ coordinate frame. Let's change the coordinate frame once more in order to simplify the computations. In this case, we want to simplify the equation of $\Pi'$ such that in the new coordinate frame ($x''y''z''$), $\Pi''$ becomes parallel to the $x''y''$-plane. For this transformation, another change of variables matrix $M$ is needed so that $X' = MX''$ for $X'' = \begin{bmatrix} x'' & y'' & z'' \end{bmatrix}^T$. Note that, the coordinate transformation matrix is just the transpose of $M$, i.e. $X'' = M^T X'$ ($M^{-1} = M^T$ for an orthogonal matrix). The desired change of variables matrix should transform the vector $(0, 0, 1)$ to $(a, b, c)$. There exists infinitely many different rotation matrices for this purpose. One of them is selected as given in the *Equation* 7.

$$M = \begin{bmatrix} \frac{-b}{\sqrt{a^2+b^2}} & \frac{-ac}{\sqrt{a^2+b^2}} & a \\ \frac{a}{\sqrt{a^2+b^2}} & \frac{-bc}{\sqrt{a^2+b^2}} & b \\ 0 & \sqrt{a^2+b^2} & c \end{bmatrix} \tag{7}$$

Using $M$, the change of variables yields the plane and the cone in the $x''y''z''$ coordinate frame as given in *Equations* 8 and 9 respectively.

$$\Pi'' : z'' + d = 0 \tag{8}$$

$$\begin{aligned} &\Phi'' : (MX'')^T D(MX'') = X''^T M^T D M X'' = \\ &\lambda_1 \left( -\frac{acy''}{\sqrt{a^2+b^2}} + az'' - \frac{bx''}{\sqrt{a^2+b^2}} \right)^2 + \\ &\lambda_2 \left( \frac{ax''}{\sqrt{a^2+b^2}} - \frac{bcy''}{\sqrt{a^2+b^2}} + bz'' \right)^2 + \\ &\lambda_3 \left( cz'' + y''\sqrt{a^2+b^2} \right)^2 = 0 \end{aligned} \tag{9}$$

The intersection of $\Phi''$ and $\Pi''$ can be computed in the $x''y''z''$ coordinate frame and the conditions under which the resultant conic (*Equation* 10) is a 3D circle can be found. Note that, the transformations that are carried out are all rigid, i.e. the shape and size of the objects are not altered. Therefore, the identified relationships in the $x''y''z''$ coordinate frame can be reverted back to the $xyz$ coordinate frame.

$$\begin{aligned}&\left[\frac{\lambda_1 b^2}{a^2+b^2}+\frac{\lambda_2 a^2}{a^2+b^2}\right]x''^2+\left[\frac{2\lambda_1 abc}{a^2+b^2}-\frac{2\lambda_2 abc}{a^2+b^2}\right]x''y''+\\ &\left[\frac{\lambda_1 a^2c^2}{a^2+b^2}+\frac{\lambda_2 b^2c^2}{a^2+b^2}+\lambda_3(a^2+b^2)\right]y''^2-d\left[-\frac{2\lambda_1 ab}{\sqrt{a^2+b^2}}+\frac{2\lambda_2 ab}{\sqrt{a^2+b^2}}\right]x''+\\ &-d\left[-\frac{2\lambda_1 a^2c}{\sqrt{a^2+b^2}}-\frac{2\lambda_2 b^2c}{\sqrt{a^2+b^2}}+2\lambda_3 c\sqrt{a^2+b^2}\right]y''+\left[\lambda_1 a^2+\lambda_2 b^2+\lambda_3 c^2\right]d^2=0\end{aligned} \tag{10}$$

In order the *Equation* 10 to be a circle[6], the conditions given in *Equations* 11, 12 and 13 must hold.

$$(\lambda_1-\lambda_2)\,(abc)=0 \tag{11}$$

$$a^2+b^2\neq 0 \tag{12}$$

$$\lambda_1 b^2+\lambda_2 a^2=c^2\left(\lambda_1 a^2+\lambda_2 b^2\right)+\lambda_3\left(a^2+b^2\right)^2 \tag{13}$$

*Equation* 11 holds if $\lambda_1=\lambda_2$ and / or the product $abc=0$. The former leads to a special case which will be discussed soon. The later requires at least one of $a$, $b$ or $c$ to be zero but according to *Equation* 12 both $a$ and $b$ cannot be zero. Also remember the assumptions made earlier: $a^2+b^2+c^2=1$, $\lambda_1>0$, $\lambda_2>0$ and $\lambda_3<0$. Let's analyse the remaining condition (*Equation* 13) along with these constraints under three different cases (cases I, II, and III) and discuss the special case with an additional case (case IV).

**CASE-I:** $a=0$

This yields four possible solutions for $(a,b,c)$ under the constraint $\lambda_2>\lambda_1$ (*Equation* 14).

$$a=0,\quad b=\pm\sqrt{\frac{\lambda_2-\lambda_1}{\lambda_2-\lambda_3}},\quad c=\pm\sqrt{\frac{\lambda_1-\lambda_3}{\lambda_2-\lambda_3}} \tag{14}$$

**CASE-II:** $b=0$

This yields four possible solutions for $(a,b,c)$ under the constraint $\lambda_1>\lambda_2$ (*Equation* 15).

$$a=\pm\sqrt{\frac{\lambda_1-\lambda_2}{\lambda_1-\lambda_3}},\quad b=0,\quad c=\pm\sqrt{\frac{\lambda_2-\lambda_3}{\lambda_1-\lambda_3}} \tag{15}$$

[6] A conic equation $ax^2+bxy+cy^2+dx+ey+f=0$ represents a circle if and only if $a=c$ and $b=0$. Then, the circle centre and radius can be computed as $(-d/2a,-e/2a)$ and $\sqrt{d^2+e^2-4af}/2|a|$ respectively.

**CASE-III:** $c = 0$

This yields four possible solutions for $(a, b, c)$ under the constraints $\lambda_1 > \lambda_2$ and $\lambda_2 > \lambda_1$ (*Equation* 16). Observe that, the constraints contradict with each other, thus no solution is obtained from this case.

$$a = \pm\sqrt{\frac{\lambda_1 - \lambda_3}{\lambda_1 - \lambda_2}}, \quad b = \pm\sqrt{\frac{\lambda_2 - \lambda_3}{\lambda_2 - \lambda_1}}, \quad c = 0 \tag{16}$$

**CASE-IV:** $\lambda_1 = \lambda_2 = \lambda$

For this case, *Equation* 13 becomes $(\lambda - \lambda_3)(a^2 + b^2)^2 = 0$. Observe that the two solutions of this equation are $\lambda = \lambda_3$ and $a = b = 0$ which contradict with the already made assumptions. However, notice that for the $\lambda_1 = \lambda_2$ case, we do not need to perform this analysis in the $x''y''z''$ coordinate frame since the *Equation* 5 becomes a right circular cone in the $x'y'z'$ coordinate frame. That is, any plane parallel to the $x'y'$ plane intersects with the cone $(\Phi')$ in circles. For instance, if we choose $\Pi'$ as $z' = \beta$, then $\Pi'$ and $\Phi'$ intersect on a 3D circle $(\mathcal{C}')$ whose centre $(\rho')$, normal $(n')$ and radius $(r)$ can be computed as given in the *Equation* 17.

$$\rho' \to (0, 0, \beta), \quad n' \to (0, 0, 1), \quad r = |\beta|\sqrt{\frac{-\lambda_3}{\lambda}} \tag{17}$$

For a unit circle, $\beta = \pm\sqrt{-\lambda/\lambda_3}$. Observe that, the two different values of $\beta$ generate two symmetric planes that intersect with the cone $(\Phi')$ and yield two different 3D circles which have the same normal and radius but symmetric origins w.r.t. the origin of the $x'y'z'$ coordinate frame. The corresponding 3D circles in the camera coordinate frame can be obtained by applying the backward transformations (*Equation* 18). The symmetry of the centres are preserved in the camera coordinate frame and one of the two solutions can be eliminated according to the viewing direction of the camera. The camera is looking down the negative $z$-axis, thus the 3D circle that lies on the negative part of the $z$-axis becomes the only solution.

$$\rho = P\rho' = \beta e_3, \quad n = Pn' = e_3 \tag{18}$$

For ***CASE-I*** and ***CASE-II***, we have not found the intersected 3D circles yet. One can construct the *Equation* 10 for different solutions of $n' = (a, b, c)$ (see *Equations* 14 and 15) and compute the corresponding centres $(\rho'')$ and radii $(r)$. We will perform these computations only for ***CASE-I*** since the computations for ***CASE-II*** are exactly same other than the parameter values.

For ***CASE-I***, the radius and the corresponding $d$ (the signed distance between the origin of the $x'y'z'$ coordinate frame and the plane $\Pi'$) are computed as in the *Equation* 19.

$$r = \frac{|d|}{\lambda_1}\sqrt{-\lambda_2\lambda_3} \quad and\ d = \pm\frac{\lambda_1}{\sqrt{-\lambda_2\lambda_3}}\ for\ r = 1 \tag{19}$$

Eight different planes $(\Pi')$ can be constructed based on the possible values of $n' = (a, b, c)$ and $d$. *Equation* 20 displays all of the possible circle centres and normals. Note that the centres are in the $x''y''z''$ coordinate frame whereas the normals are in the $x'y'z'$ coordinate frame.

$$\begin{aligned}
&\rho''_1 = |d|\begin{pmatrix}0 & -\delta & -1\end{pmatrix} && for\ \ n'_1 = \begin{pmatrix}0 & |b| & |c|\end{pmatrix} && and\ \ d > 0\\
&\rho''_2 = |d|\begin{pmatrix}0 & \delta & 1\end{pmatrix} && for\ \ n'_2 = \begin{pmatrix}0 & |b| & |c|\end{pmatrix} && and\ \ d < 0\\
&\rho''_3 = |d|\begin{pmatrix}0 & \delta & -1\end{pmatrix} && for\ \ n'_3 = \begin{pmatrix}0 & |b| & -|c|\end{pmatrix} && and\ \ d > 0\\
&\rho''_4 = |d|\begin{pmatrix}0 & -\delta & 1\end{pmatrix} && for\ \ n'_4 = \begin{pmatrix}0 & |b| & -|c|\end{pmatrix} && and\ \ d < 0\\
&\rho''_5 = |d|\begin{pmatrix}0 & -\delta & -1\end{pmatrix} && for\ \ n'_5 = \begin{pmatrix}0 & -|b| & |c|\end{pmatrix} && and\ \ d > 0\\
&\rho''_6 = |d|\begin{pmatrix}0 & \delta & 1\end{pmatrix} && for\ \ n'_6 = \begin{pmatrix}0 & -|b|, & |c|\end{pmatrix} && and\ \ d < 0\\
&\rho''_7 = |d|\begin{pmatrix}0 & \delta & -1\end{pmatrix} && for\ \ n'_7 = \begin{pmatrix}0 & -|b| & -|c|\end{pmatrix} && and\ \ d > 0\\
&\rho''_8 = |d|\begin{pmatrix}0 & -\delta & 1\end{pmatrix} && for\ \ n'_8 = \begin{pmatrix}0 & -|b| & -|c|\end{pmatrix} && and\ \ d < 0
\end{aligned} \tag{20}$$

where

$$\delta = \frac{\sqrt{(\lambda_2 - \lambda_1)(\lambda_1 - \lambda_3)}}{\lambda_1}$$

Observe in *Equation* 20 that $\rho''_1 = \rho''_5$, $\rho''_2 = \rho''_6$, $\rho''_3 = \rho''_7$ and $\rho''_4 = \rho''_8$. In fact, all these circles have the same normal (observe that supporting planes have the same normal $n'' = (0, 0, 1)$) in the $x''y''z''$ coordinate frame which is enough to conclude there will be four different solutions in the camera coordinate frame. However, we will do further analysis for completeness. Circle centres in $x'y'z'$ coordinate frame can be computed by $\rho' = M\rho''$. Observe in *Equation* 7 that, the matrix $M$ depends on the values of the $a$, $b$ and $c$ and it can be simplified for ***CASE-I*** (*Equation* 21).

$$M = \begin{bmatrix} -sign(b) & 0 & 0 \\ 0 & -sign(b)sign(c)|c| & sign(b)|b| \\ 0 & |b| & sign(c)|c| \end{bmatrix} \tag{21}$$

The obtained centres in the $x'y'z'$ coordinate frame are displayed in the *Equation* 22. Observe that, $\rho'_1 = \rho'_8$, $n'_1 = -n'_8$, $\rho'_2 = \rho'_7$, $n'_2 = -n'_7$, $\rho'_3 = \rho'_6$, $n'_3 = -n'_6$, $\rho'_4 = \rho'_5$ and $n'_4 = -n'_5$. Notice that, the circles which have identical centres have normals in the opposite directions (symmetric normals w.r.t origin). This is the result of using both positive and negative values of $d$. As a result, once the sign of $d$ is chosen, four solutions are obtained and reversing the sign of $d$, alters the direction of the computed circle normals. The centre locations are not affected by the sign of the $d$. Furthermore, notice also that $\rho'_1$ and $\rho'_7$, $\rho'_2$ and $\rho'_8$, $\rho'_3$ and $\rho'_5$, $\rho'_4$ and $\rho'_6$ are symmetric w.r.t the origin of the $x'y'z'$

coordinate frame.

$$\begin{aligned}
\rho'_1 &= |d| \begin{pmatrix} 0 & \delta|c| - |b| & -\delta|b| - |c| \end{pmatrix} \\
\rho'_2 &= -\rho'_1 \\
\rho'_3 &= |d| \begin{pmatrix} 0 & \delta|c| - |b| & \delta|b| + |c| \end{pmatrix} \\
\rho'_4 &= -\rho'_3 \\
\rho'_5 &= |d| \begin{pmatrix} 0 & -\delta|c| + |b| & -\delta|b| - |c| \end{pmatrix} \\
\rho'_6 &= -\rho'_5 \\
\rho'_7 &= |d| \begin{pmatrix} 0 & -\delta|c| + |b| & \delta|b| + |c| \end{pmatrix} \\
\rho'_8 &= -\rho'_7
\end{aligned} \tag{22}$$

The normals and centres of the 3D circles can now be transformed back to the camera coordinate frame by $\rho = P\rho' \quad n = Pn'$. Then, two of the four 3D circles can be eliminated based on the camera's viewing direction and the symmetry between the computed circles. As a result, the remaining two circles that lie on the negative side of the $z$-axis are accepted as the solutions. These circles can now be scaled to desired depth or radius using the similar triangles ($\rho_2 = (r_1/r_2)\rho_1$). *Figures* 7.6 and 7.7 display 3D circles that are computed using the methods explained in this section.

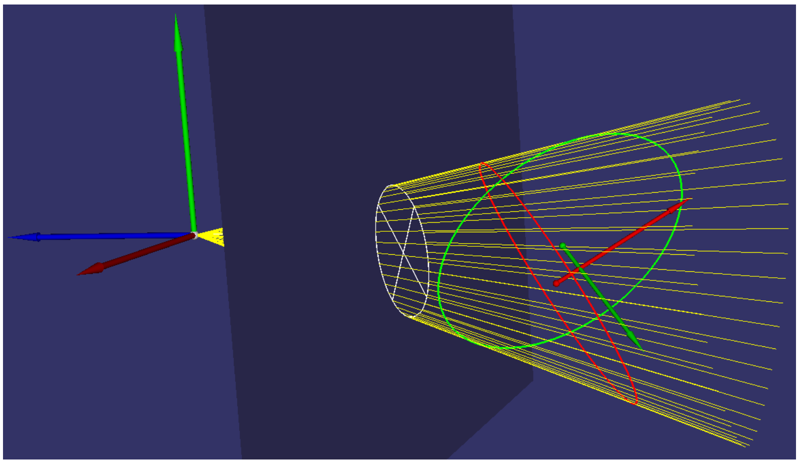


**Figure 7.6:** Computed 3D circles from a perspective projection: Coordinate frame (XYZ → RGB), image plane (dark blue) and the projected ellipse (white) on the image plane, two 3D circles (red and green) and perspective projection lines (yellow).

## DEGENERATE CASE:

The only remaining analysis that has to be done is related to the case when the projection of the 3D circle is not an ellipse. A 3D circle's projection is a line segment when the plane of the circle passes through the centre of projection. Let $p_1$ and $p_2$ be the endpoints of the line segment on the image plane. Then, notice that the normal of the 3D circle can be computed as $\vec{n} = \vec{p_1} \times \vec{p_2}$ and the circle centre has the same direction with the $\vec{p_1} + \vec{p_2}$ vector. After some arithmetic based on the *Figure* 7.8, one can find $r = ||\vec{\rho}||\, sin(\alpha)$.

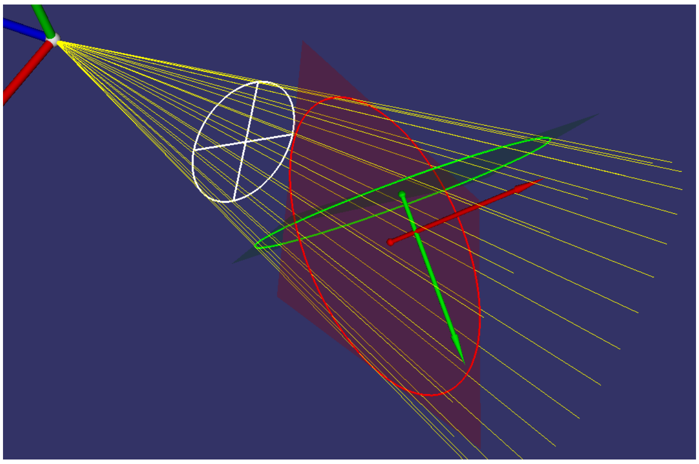


**Figure 7.7:** The same circles in the *Figure* 7.6 are displayed with a different point of view. This time, the image plane is not displayed but the supporting planes are added.

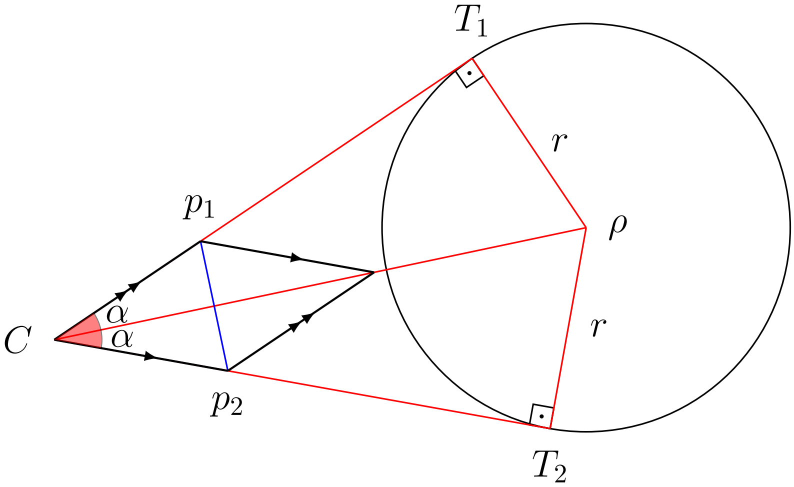


**Figure 7.8:** 3D circle projection for the degenerate case: centre of projection ($C$), circle centre ($\rho$), projection of the circle (line segment between the points $p_1$ and $p_2$), perspective projection lines passing through $p_1$ and $p_2$ are tangent to the circle at $T_1$ and $T_2$. The centre of the circle has the same direction with the vector $p_1 + p_2$.

As a result, for a fixed depth or radius and a specific viewing direction, this problem has a single solution for the special cases (when the projection of the 3D circle is a circle or a line segment) and two solutions for the general case (the projection of the 3D circle is an ellipse).

#### 7.2.4.2 Estimating 3D Circles Under Orthographic Projection

Under orthographic projection, 3D circles are also projected as ellipses other than the special cases. If the supporting plane of the 3D circle is parallel to the image plane, then the circle is projected as a circle. On the other hand, if supporting plane and the image plane are perpendicular to each other, then the projection will be a line segment whose length is equal to the diameter of the circle.

The ratio of lengths are preserved under orthographic projection which leads to the following property: the centre of a 3D circle is projected as the centre of the projected ellipse and any line segment passing through the ellipse centre whose endpoints are on the ellipse is the projection of a circle diameter. Therefore, when the circle centre is positioned on the image plane, the centres of the projected ellipse and the circle becomes the same point and the major axis of the ellipse becomes an actual diameter of the 3D circle. Consequently, given an ellipse ($\mathcal{E}$) on the image plane, one can construct a 3D circle ($\mathcal{C}_i$) on the image plane whose centre and diameter coincide with the ellipse centre and major axis, respectively. Then, the estimation problem can be reformulated as follows: find the tilt angle ($\theta$), that is applied to $\mathcal{C}_i$ so that the tilted 3D circle ($\mathcal{C}_t$) orthographically projects to $\mathcal{E}$.

The semi-major and semi-minor axes of the projected ellipse are represented with 3D vectors pointing outward from the ellipse centre. There exist four different orientations of these vectors (*Figure* 7.9). We will see how the orientation of these vectors change the estimated circles. For our analysis, we select the orientation (a) given in the *Figure* 7.9.

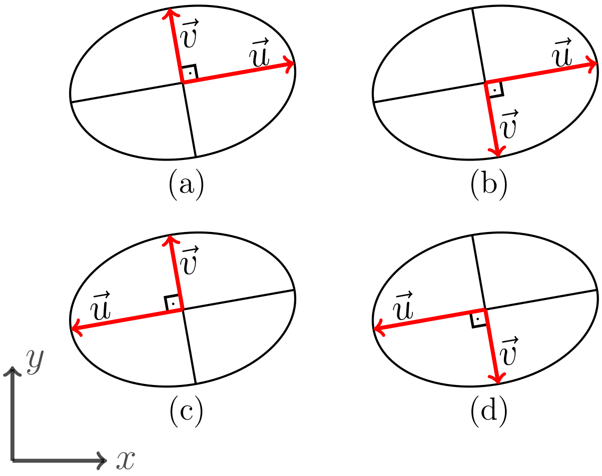


**Figure 7.9:** Four possible orientations of the semi-major and the semi-minor axes of an ellipse: semi-major vector ($\vec{u}$) and semi-minor vector ($\vec{v}$).

*Figure* 7.10 displays the concentric 3D circle ($\mathcal{C}_i$) and the ellipse ($\mathcal{E}$) on the image plane. According to the reformulation of the problem, we want to rotate $\mathcal{C}_i$ around its diameter that coincides with the ellipse's major axis.

The normal ($n_i$) of $\mathcal{C}_i$ is taken in the direction of the cross product $\vec{u} \times \vec{v}$. Therefore, $n_i = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$ for our selected orientation. In addition, $\vec{u}$ and $\vec{v}$ can be written in terms of the angle of rotation ($\phi$) as given in the *Equation* 23.

$$\begin{aligned} \vec{u} &= \begin{pmatrix} u_x & u_y & 0 \end{pmatrix} &= ||\vec{u}|| \begin{pmatrix} \cos\phi & \sin\phi & 0 \end{pmatrix} \\ \vec{v} &= \begin{pmatrix} v_x & v_y & 0 \end{pmatrix} &= ||\vec{v}|| \begin{pmatrix} -\sin\phi & \cos\phi & 0 \end{pmatrix} \end{aligned} \tag{23}$$

$\mathcal{C}_i$ can be tilted (rotated) either towards or away from the camera center. We fix the direction of the tilt towards the center of the camera. In order to compute the amount of tilt, a vector ($\vec{r_1}$) of length $||\vec{u}||$ that is aligned with $\vec{v}$ is utilized (*Figures* 7.10 and 7.11). Observe that, if the projection of $\vec{r_2}$ matches with $\vec{v}$ on the image plane, then the orthographic projection of $\mathcal{C}_t$ matches with the $\mathcal{E}$ (*Figure* 7.11).

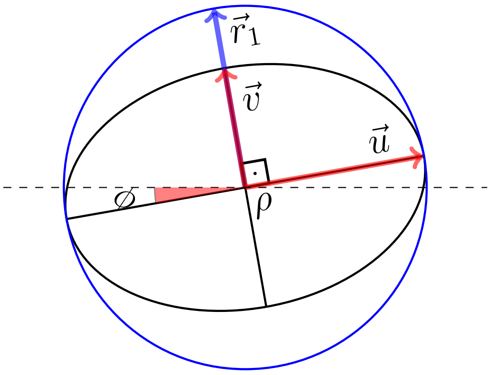


**Figure 7.10:** Concentric ellipse ($\mathcal{E}$) and circle ($\mathcal{C}_i$) on the image plane when looking from the origin of the camera coordinate frame: centre of $\mathcal{E}$ and $\mathcal{C}_i$ ($\rho$), semi-major vector ($\vec{u}$), semi-minor vector ($\vec{v}$), angle of rotation ($\phi$), a vector on the image plane ($\vec{r_1}$) that is aligned with $\vec{v}$ and its length is equal to the radius of $\mathcal{C}_i$ which is also equal to the length of $\vec{u}$. The dashed black line is the horizontal $x$-axis on the image plane.

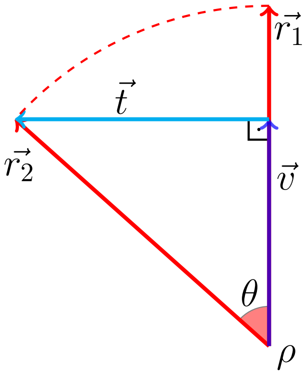


**Figure 7.11:** Computing the tilt angle ($\theta$). This figure is drawn according to a viewer looking from the tip of the semi-major vector ($\vec{u}$). That is, the semi-major vector is directed out of page. $\rho$ is the centre of the ellipse ($\mathcal{E}$) and the 3D circle ($\mathcal{C}_i$) that lie on the image plane. $\vec{v}$ is the semi-minor vector. $\vec{r_1}$ is a vector on the image plane that is aligned with $\vec{v}$ and its magnitude is equal to the radius of $\mathcal{C}_i$ (i.e. $r = ||\vec{r_1}|| = ||\vec{u}||$) and $\vec{r_2}$ is the vector obtained by rotating $\vec{r_1}$ around the semi-major vector in the counter clock-wise direction by the amount of tilt angle. $\vec{t}$ is the tilt vector perpendicular to the image plane towards the camera center. Observe that $\vec{r_2} = \vec{v} + \vec{t}$.

The tilt angle ($\theta$) and the tilt vector ($\vec{t}$ in the *Figure* 7.11) can be computed as given in *Equation* 24. Furthermore, the normal ($n_t$) of $\mathcal{C}_t$ can be obtained either by rotating $n_i$ by the tilt angle around the same axis or by the *Equation* 25.

$$\cos\theta = \frac{||\vec{v}||}{||\vec{u}||}, \quad \vec{t} = \begin{pmatrix} 0 & 0 & \sqrt{||\vec{u}||^2 - ||\vec{v}||^2} \end{pmatrix} \tag{24}$$

$$\begin{aligned} \vec{r_2} &= \vec{v} + \vec{t} \\ \vec{n_t} &= \vec{u} \times \vec{r_2} \\ &= \begin{pmatrix} \sin\theta\sin\phi & -\sin\theta\cos\phi & \cos\theta \end{pmatrix} \end{aligned} \tag{25}$$

*Figures* 7.12 and 7.13 display an estimated 3D circle from a projected ellipse under orthographic projection.

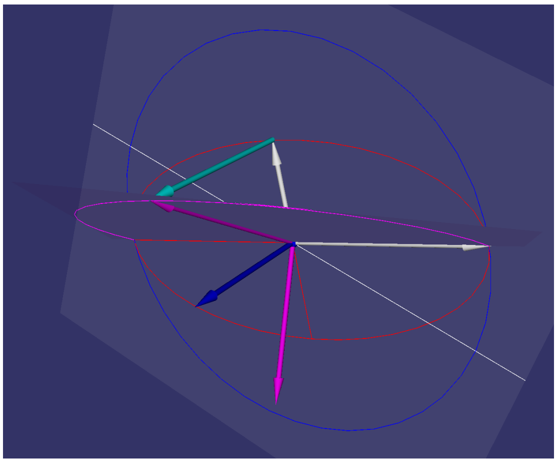


**Figure 7.12:** Estimated 3D circle under orthographic projection. The ellipse ($\mathcal{E}$) and the circle ($\mathcal{C}_i$) on the image plane are displayed in red and blue respectively. The estimated circle ($\mathcal{C}_t$) is displayed in magenta. The semi-major and semi-minor axis vectors are displayed in white and the tilt vector is displayed in cyan. Finally, the white horizontal line on the image plane indicates the $x$-axis.

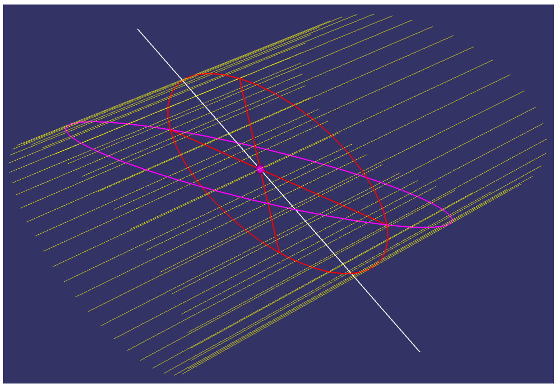


**Figure 7.13:** This figure displays a simplified version of the *Figure* 7.12. Additionally, orthographic projection lines are also displayed (yellow).

Note that the performed calculations are based on the selected orientation of the ellipse axes (*Figure* 7.9). When the same analysis is carried out for the other orientations, different solutions are obtained. *Figure* 7.14 displays the four different estimated 3D circles with four different orientations of the axes vectors. Please read the caption of the figure for the details. Recall that, we fixed the direction of the tilt vector towards the camera and select the normal of $\mathcal{C}_i$ along the direction of $\vec{u} \times \vec{v}$. Both of these vectors could have been taken in the opposite directions which would yield the same 3D circles (as displayed in *Figure* 7.14). However, just toggling the direction of the tilt or the $n_i$ still generates the same set of 3D circles but this changes the bijection between the axes vector orientations and the computed 3D circles.

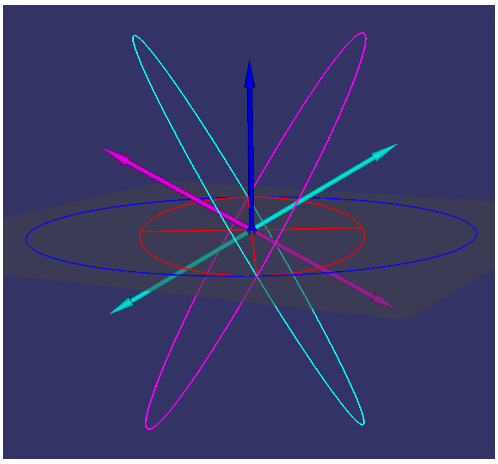


**Figure 7.14:** Four possible 3D circles estimated under orthographic projection. The projected ellipse and the circle on the image plane are displayed with red and blue, respectively. Notice that, the image plane is horizontal in this figure. *Figure* 7.9 displays the four possible orientations of the semi-major and semi-minor axes of the projected ellipse. Computations based on configuration (a) and (c) yield the 3D circles displayed in magenta whereas for (b) and (d) the cyan circles are computed. Notice that, for these pairs, the same circles are obtained but with opposite normal orientations.

#### 7.2.4.3 Further Analysis on the Perspective Projection of 3D Circles

In *Section* 7.2.4.1 we have seen the analytic method of finding the two 3D circles at a fixed depth or radius whose projections are the same given ellipse on the image plane under the perspective projection. In this section, perspective projection of 3D circles are discussed a bit further.

*Figure* 7.15 displays one of the two 3D circles on the left and their projection on the right. Notice that, the centres of the circles do not project as the centre of the ellipse and the axes of the projected ellipse do not correspond to diameters of the circles. This is due to the perspective projection in which the ratio of the lengths are not preserved (on the contrary to the orthographic projection).

Recall that in *Section* 7.2.3, we mentioned two different drawing methods of the first 2D profile which result in the same 2D ellipse. These two different drawing methods are used to select one of the two possible 3D circles that are computed using the drawn ellipse. The user's task is defined as always doing the third click towards side which is closer to the camera. According to this, the expected drawing that should be done in *Figure* 7.4 is 7.4(e). The normal and centre equations derived in *Section* 7.2.4.1 always (except the special cases) yield two 3D circles that are oppositely oriented with respect to the viewing direction.

In *Section* 7.2.1, we have discussed the work presented by Chen et al. [2013]. Recall that, in their work 3D circles are initially estimated such that the circle and the projected ellipse centres coincide. This can only be achieved by a ratio preserving projection such as orthogonal projection. The post optimization process (see *Section* 7.2.1 for a brief overview or the corresponding article for the details) utilized in their work compensates for the non-perspective construction of the generalized cylinders at the cost of deviating from 3D circles by warping them into 3D ellipses.

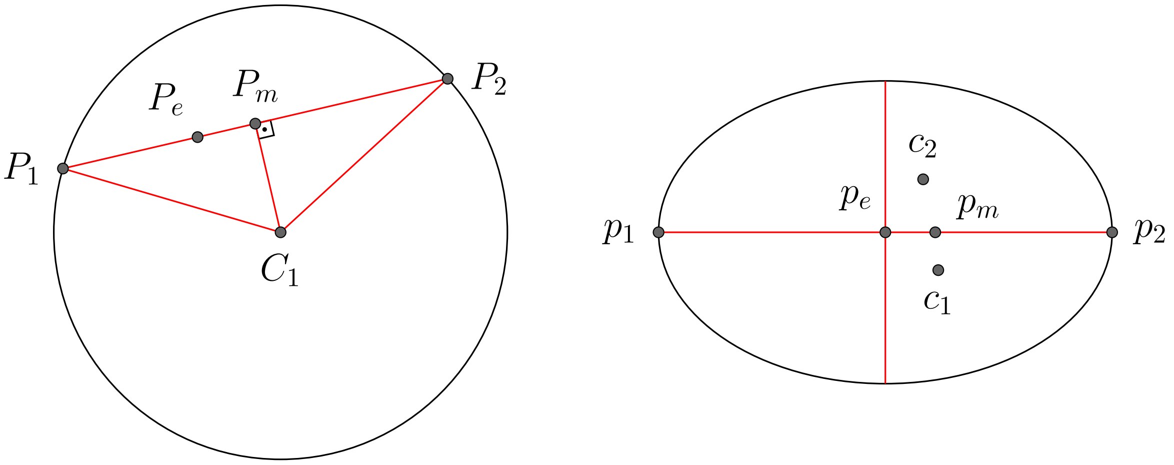


**Figure 7.15:** Perspective projection of a 3D circle. The upper-case and the lower-case labels on the circle and the ellipse indicate the corresponding points such that the lower-case labels are the perspective projections of the upper-case ones. Recall that at a given depth or fixed radius there exist two 3D circles whose projections are the same ellipse. In the figure, only one of the circles is displayed, but the projection of the second circle's centre is also displayed ($c_2$). $p_e$ is the centre of the ellipse, $p_1$ and $p_2$ are the endpoints of the major axis, $C_1$ is centre of the 3D circle and $P_m$ is midpoint of the $P_1$ and $P_2$.

In the next *Section* (7.2.5), we will derive the parametric representation of the centre of a 3D circle whose perspective projection is partially known. *Equation* 26 displays the coordinates of some critical points that are used in the related derivations which are also displayed in the *Figure* 7.15. These point coordinates are written with the assumption of a pin-hole camera model which is looking along the negative $z$-axis.

$$\begin{aligned}
p_1 &= \begin{bmatrix} x_1 & y_1 \end{bmatrix} & p_2 &= \begin{bmatrix} x_2 & y_2 \end{bmatrix} \\
P_{i1} &= \begin{bmatrix} x_1 & y_1 & n \end{bmatrix} & P_{i2} &= \begin{bmatrix} x_2 & y_2 & n \end{bmatrix} \\
P_1 &= Z_1 \begin{bmatrix} x_1/n & y_1/n & 1 \end{bmatrix} & P_2 &= Z_2 \begin{bmatrix} x_2/n & y_2/n & 1 \end{bmatrix} \\
P_e &= Z_e \begin{bmatrix} (x_1+x_2)/2n & (y_1+y_2)/2n & 1 \end{bmatrix} & P_m &= (P_1+P_2)/2 \\
p_e &= \begin{bmatrix} (x_1+x_2)/2 & (y_1+y_2)/2 \end{bmatrix}
\end{aligned} \tag{26}$$

### 7.2.5 Piecewise Sweeping the Generalized Cylinder Image and Estimating the Generalized Cylinder

After drawing the projection of the initial cross-section (*Section* 7.2.3), the user sweeps the major axis of the initial profile along the image of the generalized cylinder. However, instead of continuous sweeping used by Chen et al. [2013], we adopted a piecewise sweeping approach for simplicity. That is, we try to fit a 2D line segment to the outline of the generalized cylinder whenever the mouse button is clicked. During the sweep, a copy of the previously fit line segment is positioned to the current mouse point and its orientation is adjusted according to the bend of the axis. In order to fit these line segments to the object outlines, the edges in the input images have to be detected in advance. We used the gradient image for this purpose. The user drawn initial 2D elliptic profile and the following piecewise sweeping generate the input data for estimating the 3D model of the imaged generalized cylinders. An example of such data is displayed in *Figure* 7.16.

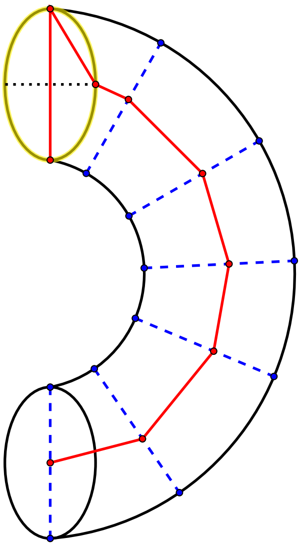


**Figure 7.16:** Generalized cylinder estimation input data: red dots are the user clicked points first three of which are used to construct the initial 2D elliptic profile (yellow), blue dashed lines illustrate the 2D line segments that are fit to the outline of the imaged generalized cylinder.

In *Section* 7.2.4.1, the computation of 3D circles from their perspective projections are presented. Notice that in the generalized cylinder modelling described in the previous paragraph, we do not have the minor axis information apart from the initial 2D elliptic profile. Therefore, the equations derived in the *Section* 7.2.4.1 cannot be used for these cases due to the insufficient input data. On the other hand, we will see that with the assumption of a known normal, a 3D circle can be computed up to a scale factor from the major axis of its projected ellipse. After deriving the necessary equations, we will continue with the discussion of the 3D generalized cylinder reconstruction.

#### 7.2.5.1 Mathematical Representation of 3D Circles

In Euclidean geometry, a 3D circle can be represented in the form a polynomial equation (as the intersection of a plane and a sphere or intersection of two spheres) or in parametric form ($C + R\cos\theta\,\vec{i} + R\sin\theta\,\vec{j}$ where $C$ is the centre of the circle, $\vec{i}$ and $\vec{j}$ are two perpendicular unit vectors that span the supporting plane of the 3D circle and $0 \leq \theta < 2\Pi$). Furthermore, one can also represent 3D circles in projective geometry using dual quadrics [Soheilian and Brédif, 2014] (*Equation* 27).

$$Q^* = R^2 \begin{bmatrix} [n]_x & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} C \\ 1 \end{bmatrix} \begin{bmatrix} C \\ 1 \end{bmatrix}^T \tag{27}$$

where $R$ is the radius of the 3D circle, $C$ is the centre and $[n]_x$ is the matrix encoding the vector product (i.e. $n \times v = [n]_x v$).

#### 7.2.5.2 Supporting Plane

The equation of the supporting plane or the plane of the 3D circle ($\Pi_0 : a_0x + b_0y + c_0z + d_0 = 0$) can be computed from $(P - P_1) \cdot \hat{n}_0 = (P - P_2) \cdot \hat{n}_0 = (P - P_e) \cdot \hat{n}_0 = 0$ where $\hat{n}_0 = \begin{bmatrix} a_0 & b_0 & c_0 \end{bmatrix}$ is the unit normal of the 3D circle and $P$ is an arbitrary point on $\Pi_0$. This yields the relations given in *Equation* 28 between the $z$-coordinates (depths) of $P_1$, $P_2$ and $P_e$.

$$Z_1 \;=\; kZ_2, \;\; Z_e \;=\; \frac{2k}{1+k}Z_2, \;\;\; where \;\; k \;=\; \frac{P_{2i} \cdot \hat{n}_0}{P_{1i} \cdot \hat{n}_0} \tag{28}$$

#### 7.2.5.3 Bisector Plane

Bisector plane ($\Pi_1 : a_1x + b_1y + c_1z + d_1 = 0$) is the plane passing through $C$ and $P_m$ and perpendicular to the supporting plane. The equation of the bisector plane can be obtained by $(C - P_m) \cdot (P_1 - P_2) = 0$ which yields the *Equation* 29.

$$\begin{aligned} n_1 &= \begin{bmatrix} a_1 & b_1 & c_1 \end{bmatrix} = \hat{n}_0 \times (P_{2i} \times P_{1i}) \\ d_1 &= Z_2 \frac{(\hat{n}_0 \cdot P_{2i})^2 (P_{1i} \cdot P_{1i}) - (\hat{n}_0 \cdot P_{1i})^2 (P_{2i} \cdot P_{2i})}{2n(\hat{n}_0 \cdot P_{1i})} \end{aligned} \tag{29}$$

Notice that the bisector plane is the locus of the points whose distance to $P_1$ and $P_2$ are equivalent. One can compute the same plane by intersecting two spheres centred at $P_1$ and $P_2$ with radii $R$ and computing the supporting plane for the intersection. That is $S_1 : ||P_1 - C||^2 = R^2$, $S_2 : ||P_2 - C||^2 = R^2$ and $\Pi_1$ is the supporting plane for the circle $S_1 \cap S_2$.

#### 7.2.5.4 Offset Planes

Observe the two lines ($l_{p_i}$) in the *Figure* 7.17 that are passing through $p_i$ and perpendicular to the major axis. Notice that, the planes ($T_i$) passing through the camera center and these lines are tangent to the 3D circle at the points $P_i$. In 2D projective space, any point $p$ on a line $l$ holds the equation: $p^T l = l^T p = 0$. Similarly, in 3D projective space, any point $P$ on a plane $\Pi$ holds the equation: $P^T \Pi = \Pi^T P = 0$. Furthermore, a 3D point is projected onto the image plane as $p = P_{mat}P$ where $P_{mat}$ is the $3 \times 4$ camera projection matrix[7]. Based on these, ${p_i}^T l_{p_i} = P_i^T {P_{mat}}^T l_{p_i} = 0$ and therefore $T_i = {P_{mat}}^T l_{p_i}$.

The tangent planes $T_i$ to the 3D circle ($Q^*$, *Equation* 27) at points $P_i$ must hold the relation between a dual quadric and its tangent planes. That is: ${T_i}^T Q^* T_i = 0$. When this expression is simplified, one can obtain $[C^T W_i]^2 + R^2({W_i}^T {[\hat{n}_0]_x}^2 W_i) = 0$ where $W_i$ is the unit normal of the tangent plane $T_i$. This equation can be decomposed into two plane equations as given in the *Equation* 30.

$$\left(C^T W_i + R\sqrt{-{W_i}^T {[\hat{n}_0]_x}^2 W_i}\right)\left(C^T W_i - R\sqrt{-{W_i}^T {[\hat{n}_0]_x}^2 W_i}\right) = 0 \tag{30}$$

[7] $P_{mat} = KR[I_3| - S]$ where $K$ is the calibration matrix of the camera, $R$ is the orientation of the camera, $I_3$ is the $3 \times 3$ identity matrix and $S$ is the position of the camera.

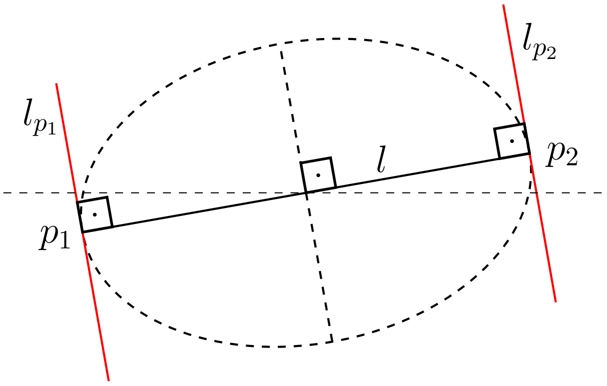


**Figure 7.17:** Computing the tangent lines ($l_{p_1}$ and $l_{p_2}$) to the ellipse at the major axis endpoints ($p1 = (x_1, y_1, 1)$ and $p2 = (x_2, y_2, 1)$). The major axis line ($l$) can be computed as: $l = p_1 \times p_2 = (l_1, l_2, l_3)$ and the orthogonal lines to $l$ passing through $p_1$ and $p_2$ can be computed as: $l_{p_1} = (l_2, -l_1, l_1 y_1 - l_2 x_1)$ and $l_{p_2} = (l_2, -l_1, l_1 y_2 - l_2 x_2)$.

Notice that these two planes have the same normal with the corresponding tangent plane (i.e. parallel to the corresponding tangent plane) and in fact they are the offset versions of the tangent planes. The signed distance between the corresponding tangent plane and these offset planes are $\pm R\sqrt{-W_i{}^T[\hat{n}_0]_x{}^2 W_i}$. Moreover, the signed distance between the circle centre and the tangent plane $T_i$ is $C^T W_i$ which is equal to one of the signed distances between the tangent and offset planes. Therefore, the circle centre must lie on of the offset planes. The offset plane on which the circle centre lies can be found with a very simple test which is illustrated in the *Figure* 7.18. After eliminating two of the four offset planes, two planes ($\Pi_2 : a_2 x + b_2 y + c_2 z + d_2 = 0$ and $\Pi_3 : a_3 x + b_3 y + c_3 z + d_3 = 0$) remain on which the circle centre lies.

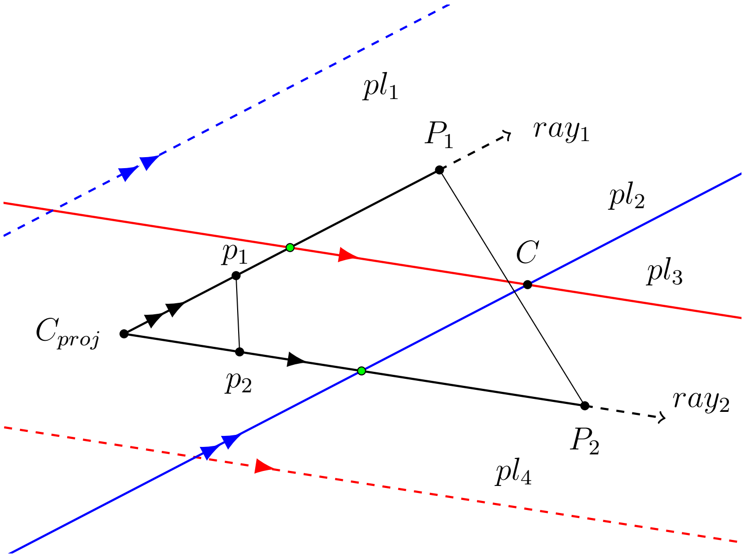


**Figure 7.18:** Eliminating offset planes: two of the four offset planes are eliminated based on the intersection test illustrated in the figure (a 2D cross-section from the top-view). Two rays ($ray_1$ and $ray_2$) are shot from the centre of projection (camera centre, $C_{proj}$) passing through the two end-points ($p_1$, $p_2$) and the offset planes that are not intersected by the rays are eliminated (Eliminated planes are displayed as dashed lines).

#### 7.2.5.5 Parametric Solution to 3D Circle Centre

Three planes ($\Pi_1$, $\Pi_2$ and $\Pi_3$) have been identified on which the centre of the 3D circle lies. The value of $d_1$ depends on $Z_2$ (also $Z_1$ and $Z_e$ via *Equation* 28) and the values of $d_2$ and $d_3$ depend on the radius of the 3D circle. Therefore, the intersection of these three planes generates a parametric solution (using Cramer's rule) to the circle centre in which the parameters are $Z_2$ and $R$. (*Equation* 31).

$$C(Z_2, R) = \begin{bmatrix} X = f_x(Z_2, R) \\ Y = f_y(Z_2, R) \\ Z = f_z(R) \end{bmatrix} = \begin{bmatrix} s_1 R + s_2 Z_2 \\ s_3 R + s_4 Z_2 \\ s_5 R \end{bmatrix} \tag{31}$$

Observe that the distance between the circle centre and $P_2$ is equal to $R$. Let's choose the scale of the circle such that it becomes a unit circle (i.e. let's take $R = 1$). Therefore, the equation $||C - P_2|| = 1$ can be solved for $Z_2$ which leads to the computation of the unit circle centre using the parametric centre equation.

As a result, from the given major axis and the normal of the 3D circle, the position of the circle can be computed upto a scale. Additionally if a radius or depth is given, then the computed unit circle can be scaled to find the 3D circle. Having found the unit circle, we can proceed for the discussion of the generalized cylinder modelling.

#### 7.2.5.6 Generalized Cylinder Modelling

The parametric centre equation (*Equation* 31) requires the circle normal which has to be estimated in order to compute the centre of the circle (up to a scale) by only using the major axis of its projection. Recall that Chen et al. [2013] made two fundamental assumptions for the same purpose: planar axis whose supporting plane is parallel to the image plane and a ratio preserving projection. These assumptions are sufficient to estimate the circles up to a scale (the scale is selected such that the circle centres lie on the image plane) from the major axes. The radii of the circles are taken as the semi-major axis lengths, the normals are taken along the user drawn sweep curve and the centres are taken as the midpoint of the major axes (i.e. ellipse centres on the image plane). Although, these assumptions generate 3D circles whose perspective projections approximate[8] the outline of the imaged generalized cylinder, their 3D positions are not correct. However, they solved the overall model (which may be composed of several separate parts) by minimizing a set of geometrically inferred constraints (geo-semantic constraints and part based internal constraints) that fixes the 3D positions of the estimated circles (the shapes of the circles can also be changed and warped into ellipses after this optimization).

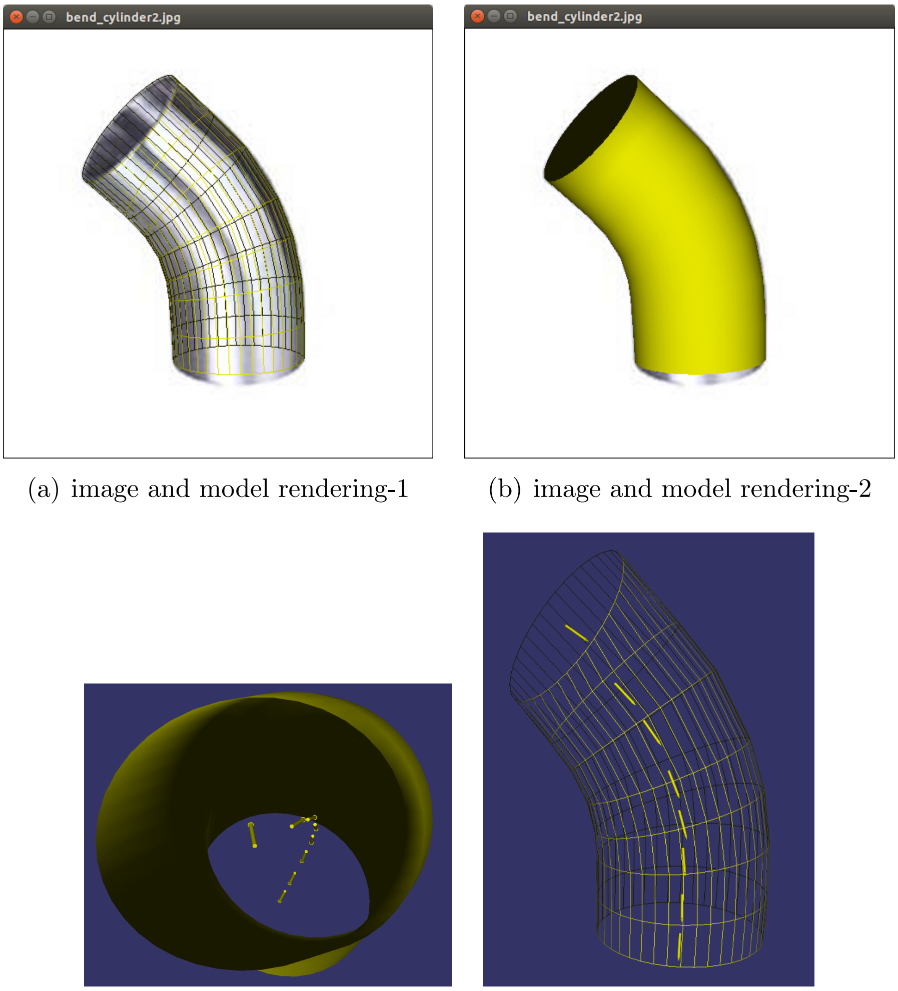
In our modelling system, we do not have an overall optimization process for fixing initially made assumptions. Therefore, we need a better way of estimating 3D circles. For the initial 3D circle we utilized the method explained in *Section* 7.2.4.1 in order to compute 3D circles from their perspective projections. However, in order to estimate the remaining 3D circles we need to infer some prior constraints. The rest of this section analyses three different prior constraints: (i) constant depth, (ii) planar axis, (iii) linear axis.

[8] The approximation errors can be seen at both ends of the generalized cylinders where either the entire or half of the projected ellipses are visible in the image.

### Constant Depth Prior:

This is the same prior used by Chen et al. [2013]. The assumption of planar axis whose supporting plane is parallel to the image plane results in 3D circles with the same depth (the $z$-coordinate of the circle centre for our camera set-up). We selected an arbitrary depth within the view frustum so that the model is visible on the screen.

The normals of the circles can be estimated easily using the constant depth prior. Notice that when the axis plane is parallel to the image plane and the normals are assumed to be in the axis plane, then the circle normals become parallel to the user drawn piecewise linear sweep curve segments. Using these normals and the major axes that are fit to the generalized cylinder outline, the corresponding circle centres are computed using the parametric centre equation (*Equation* 31). As a result, the corresponding generalized cylinder can be constructed by cascading the reconstructed circles in order (*Figures* 7.19(a) and 7.19(b)).



(a) image and model rendering-1 (b) image and model rendering-2

(c) model - top view (d) model - side view

**Figure 7.19:** A generalized cylinder modelled with constant depth prior: the projection of the model matches with the image except at the end of the generalized cylinder. Notice also that the normal of the first circle is not parallel to the approximated axis.

Observe in *Figures* 7.19(a) and 7.19(b) that the constructed model's perspective projection fits with the silhouette of the generalized cylinder in the image. Notice that, this fit is not exact other than the projection of the first 3D circle which is computed analytically according to the user drawn ellipse (i.e. the projection fit of the first circle is correct up to the user's accuracy of drawing its projection). The source of these errors might be the falsely estimated normals and/or falsely fit major axes.

Recognize that, the selected constant depth prior and the computation of the first 3D circle are independent from each other. That is, neither of the two are computed from the other. Therefore, the computed normal of the first 3D circle is not necessarily parallel to the approximated axis of the model which results in oblique models (*Figures* 7.19(c) and 7.19(d)). An oblique model might not be plausible for many users, since most of the real world objects of type generalize cylinder are right. In addition, the intermediately reconstructed 3D circles are not necessarily perpendicular to the axis of the object either. This is due to the fact that the user approximation to the axis (and the correspondingly estimated normals) is based on the projected ellipse centres which are close to the projected circle centres (*Figure* 7.15). The related error is very difficult to assess (since many parameters affect the projection of a 3D circle) but in practice it does not prevent reconstructing plausible models.

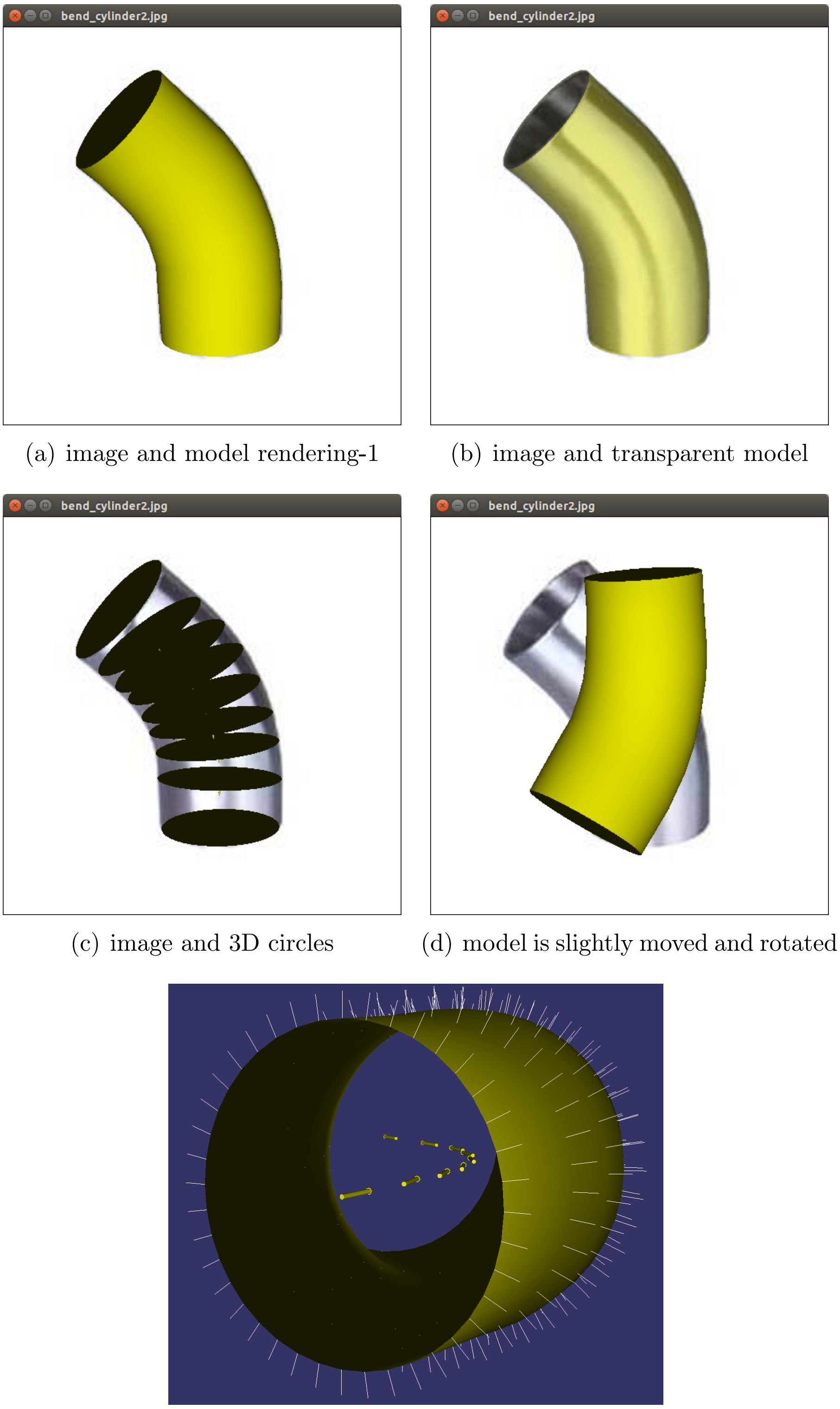
Note that the normal of the first 3D circle in the reconstructed model can be changed so that it becomes parallel to the approximated axis. However, this will increase the projection fit error and the analytical computation of the initial 3D circle becomes less useful in the first place.

**Planar Axis Prior:**

Another prior can be obtained by relaxing the parallel axis plane constraint introduced for constant depth prior. However, the input data is not sufficient to compute the plane of the axis. In order to compute the plane of the axis, it is asked from users to finish the modelling on the other end of the generalized cylinder by also drawing the projection of the last 3D circle. As a result, two circles (first and the last) upto a scale are computed analytically. A random depth or scale can be selected for one of the circles and a point (centre of the circle) on the axis plane is obtained. Then the normal of the axis plane is computed by the vector product of the two circle normals which are assumed to be two vectors on the axis plane.

Once the axis plane is defined, the normals of the intermediate 3D circles are estimated by projecting the user drawn sweep curve onto the estimated axis plane. That is, the normals are first estimated using the constant depth prior and then they are transformed to the axis plane. Having estimated the normal, the circle centres are again computed by the parametric centre equation (*Equation* 31). As a final step, the scales of circles are chosen such that the circle centres lie on the axis plane and the generalized cylinder is constructed. *Figure* 7.20 displays a modelled generalized cylinder using the same image as in *Figure* 7.19(a) but this time with planar axis prior.

When compared with the constant depth prior, the additional projection drawing mechanism also prevents the projection fit error at the end of the generalized cylinders. Therefore, the total amount of error is reduced with the planar axis prior. On the other hand, in order to draw the projection of the last 3D circle, the whole image of the generalized cylinder has to be visible in the input image. That is, partially visible generalized cylinders cannot be modelled or the user has to guess the minor axis of the last 2D profile



(a) image and model rendering-1

(b) image and transparent model

(c) image and 3D circles

(d) model is slightly moved and rotated

(e) model - top view: 3D circle and vertex normals

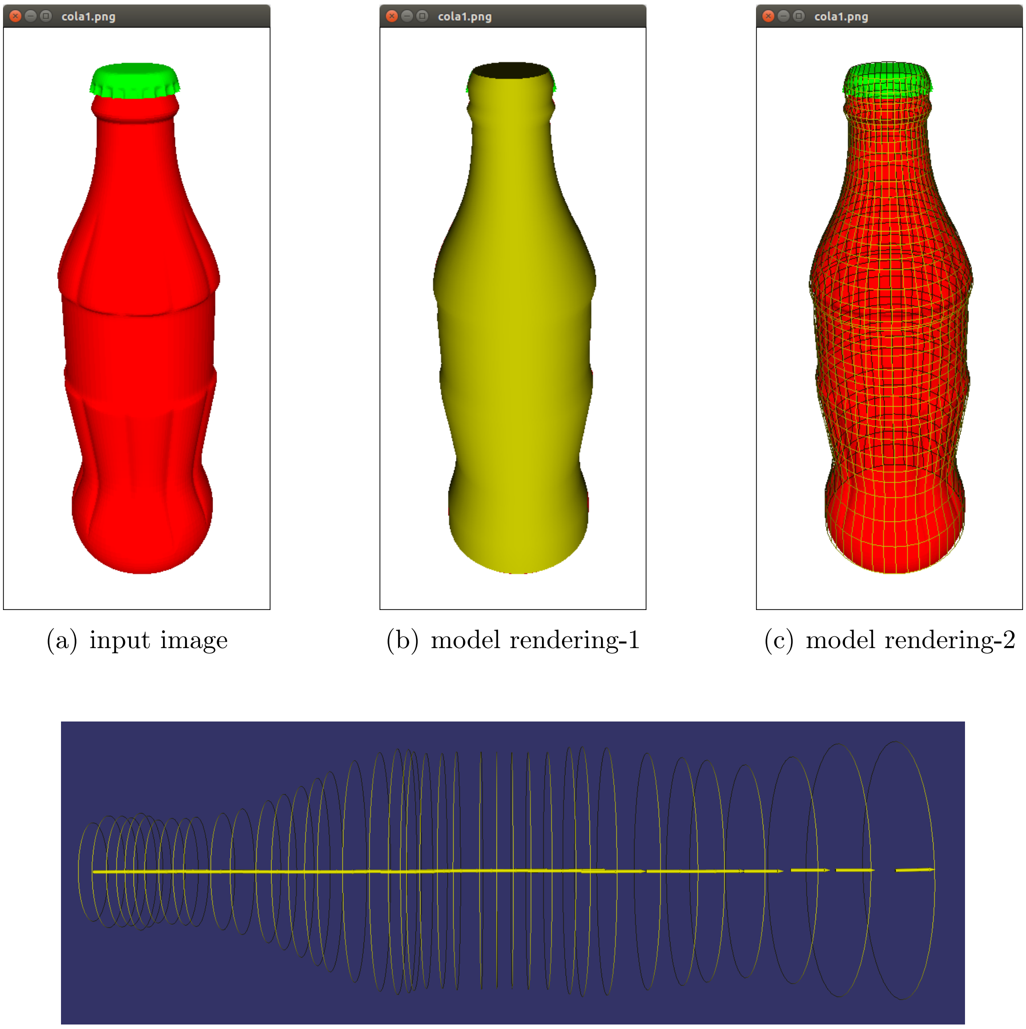
**Figure 7.20:** A generalized cylinder modelled with planar axis prior: the projection of the model matches with the image and the first circle is perpendicular to the approximated axis.

which might not be good enough to generate a plausible model. Another limitation is related to the normals of the estimated circles at both ends of the generalized cylinder.

If the two normals are parallel to each other, then the axis plane cannot be computed. A common case of the parallel normals occurs when the generalized cylinder is straight, i.e. when the normals are collinear or the axis is linear.

### Linear Axis Prior:

For straight generalized cylinders, the axis is a 3D line segment which is defined by the normal and the centre of the first 3D circle (after selecting a random depth or scale). The centre positions of the remaining circles are then computed with the parametric center (*Equation* 31) equation. The scale of these circles are selected such that the computed circle centre lies on the axis line. Note that, the rays passing through the camera centre and the circle centres may not intersect exactly with the approximated axis line (due to numerical or estimation errors). In these cases, the circle scales are selected such that their distance to the axis line is minimum. *Figure* 7.21 displays a modelled straight generalized cylinder from a synthetic image.



(a) input image (b) model rendering-1 (c) model rendering-2

(d) 3D circles of the generalized cylinder and their normals

**Figure 7.21:** A generalized cylinder modelled with linear axis prior from a synthetic image.

### More Examples and Discussions:

*Figures* 7.22, 7.23 and 7.24 display three reconstructed models from the panoramic images of the IGN's MMS (STEREOPOLIS, [Paparoditis et al., 2012]). The body of the first two models (pole and lamppost) are straight cylinders therefore modelled with linear axis prior and the tree trunk is modelled with the planar axis prior.

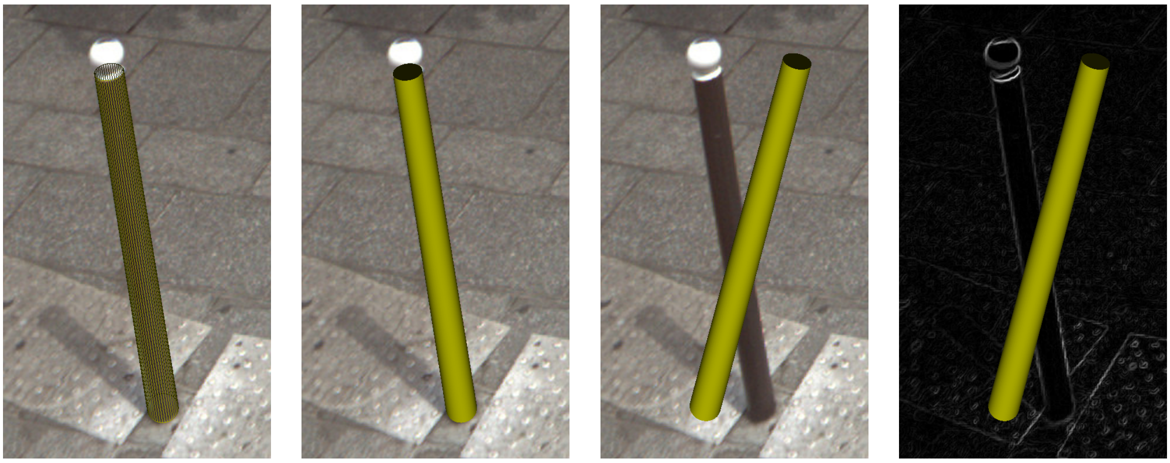


**Figure 7.22:** A modelled pole: different renderings of the model are displayed, the right most image is the gradient of the input image.

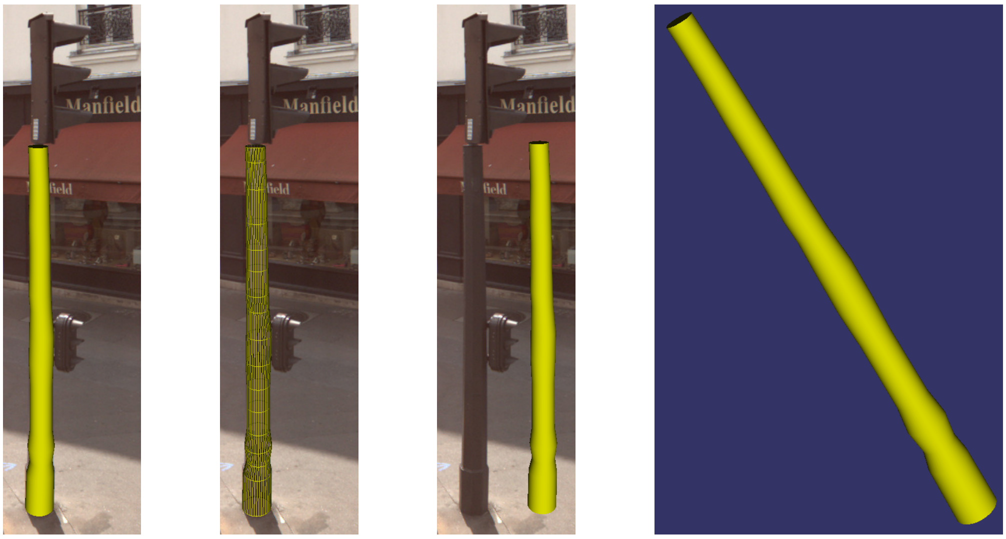
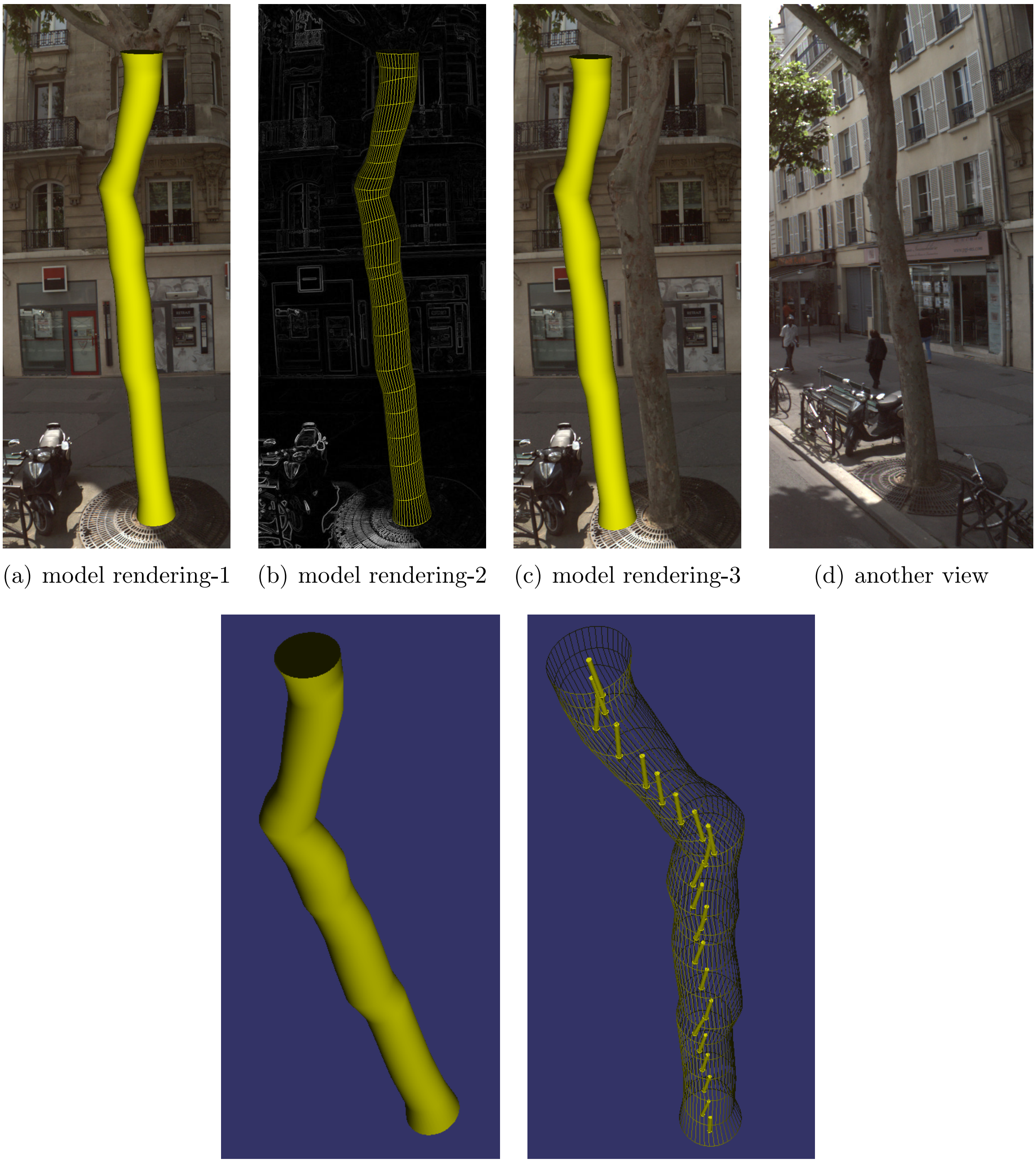


**Figure 7.23:** A modelled lamppost: different renderings of the model are displayed.

In this work, we have focused on the geometric reconstruction problem rather than the edge detection and better user interface. We use the simple gradient images in order to fit the major axes of the elliptic profiles to the outline of the generalized cylinders. Therefore, modelling from real images with a minimum error is a challenging task and it might be difficult to reconstruct a plausible model at the first trial. On the other hand, with synthetic or hand drawn images, there is not any difficulty for detecting edges which results in easier modelling.



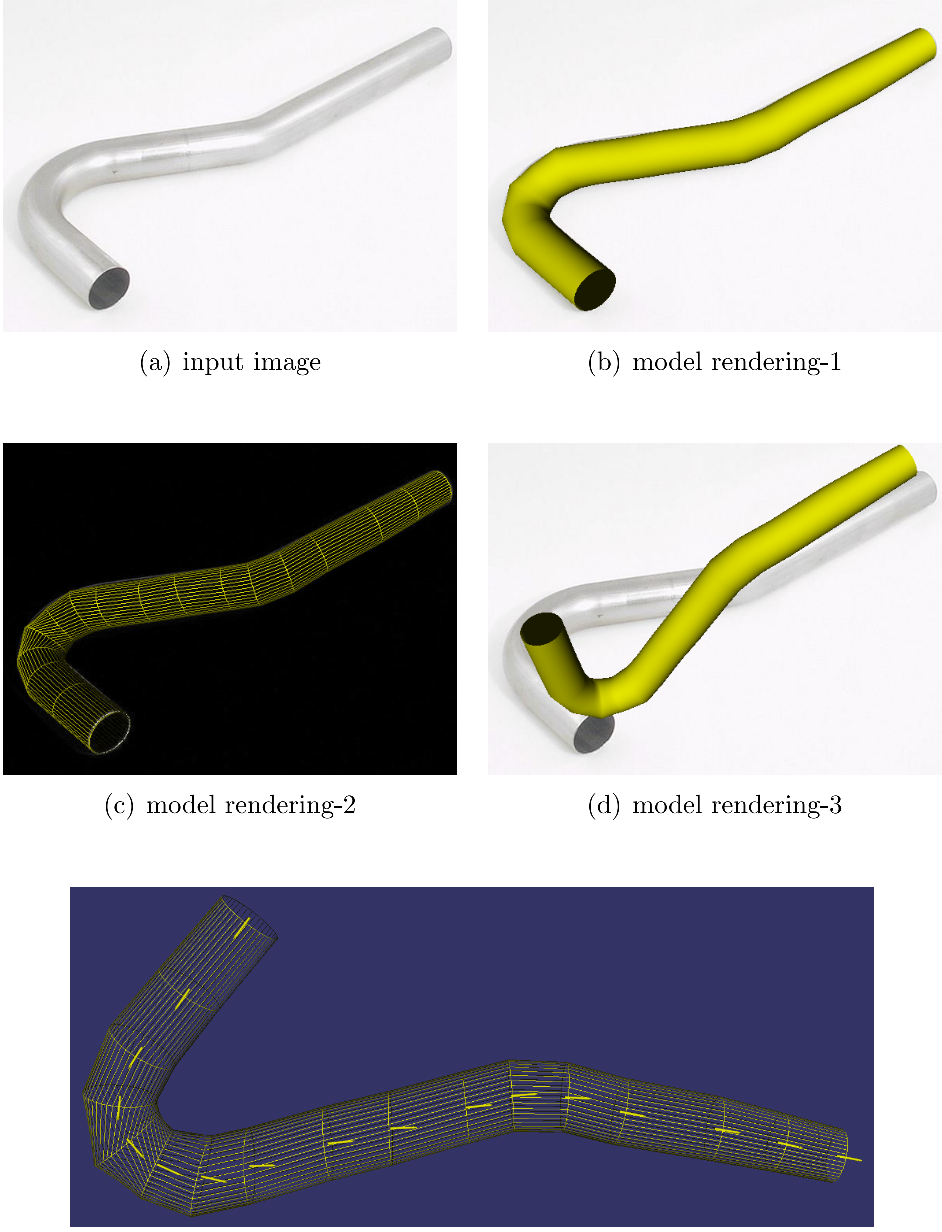
(a) model rendering-1 (b) model rendering-2 (c) model rendering-3 (d) another view

(e) model rendering-4 (f) model rendering-5

**Figure 7.24:** A 3D model of a tree trunk: (a), (b), (c), (e) and (f) display the model reconstructed from the input image and (d) displays an another image of the same tree from a different point of view which is not used in the modelling. The image at (b) is the gradient image.

In *Figure* 7.24, another view of the modelled tree trunk is displayed. When an object is modelled from a single view, the dimensions of the object along the viewing direction cannot be reconstructed (estimated). Compare the images 7.24(c) and 7.24(d) and notice the bending of the tree trunk at the upper part of it. This bending cannot be modelled from 7.24(d). As a result, multiple views of the objects are required for more accurate geometric reconstructions.

*Figure* 7.25 displays a reconstructed model of a bended pipe with the planar axis prior. Notice in this model that the radii of the reconstructed 3D circles are not equal and slightly change along the model. There might be two reasons for this noise: (i) the estimation error of the axis plane might be relatively large due to inaccurately drawn 2D elliptic profiles, (ii) the major axis fit to the outlines of the model might not be precise enough. The axis plane error can be reduced by adjusting/fitting the user drawn elliptic profiles to the edges in the images and the major axis fitting can be reduced by a better edge detection algorithm. Another solution might be defining an optimization problem (for computing the position of the circle centres) which loosens the current hard constraints (major axis fit: axis length and orientation, normal estimation) and introducing additional ones such as constant radius.



(a) input image

(b) model rendering-1

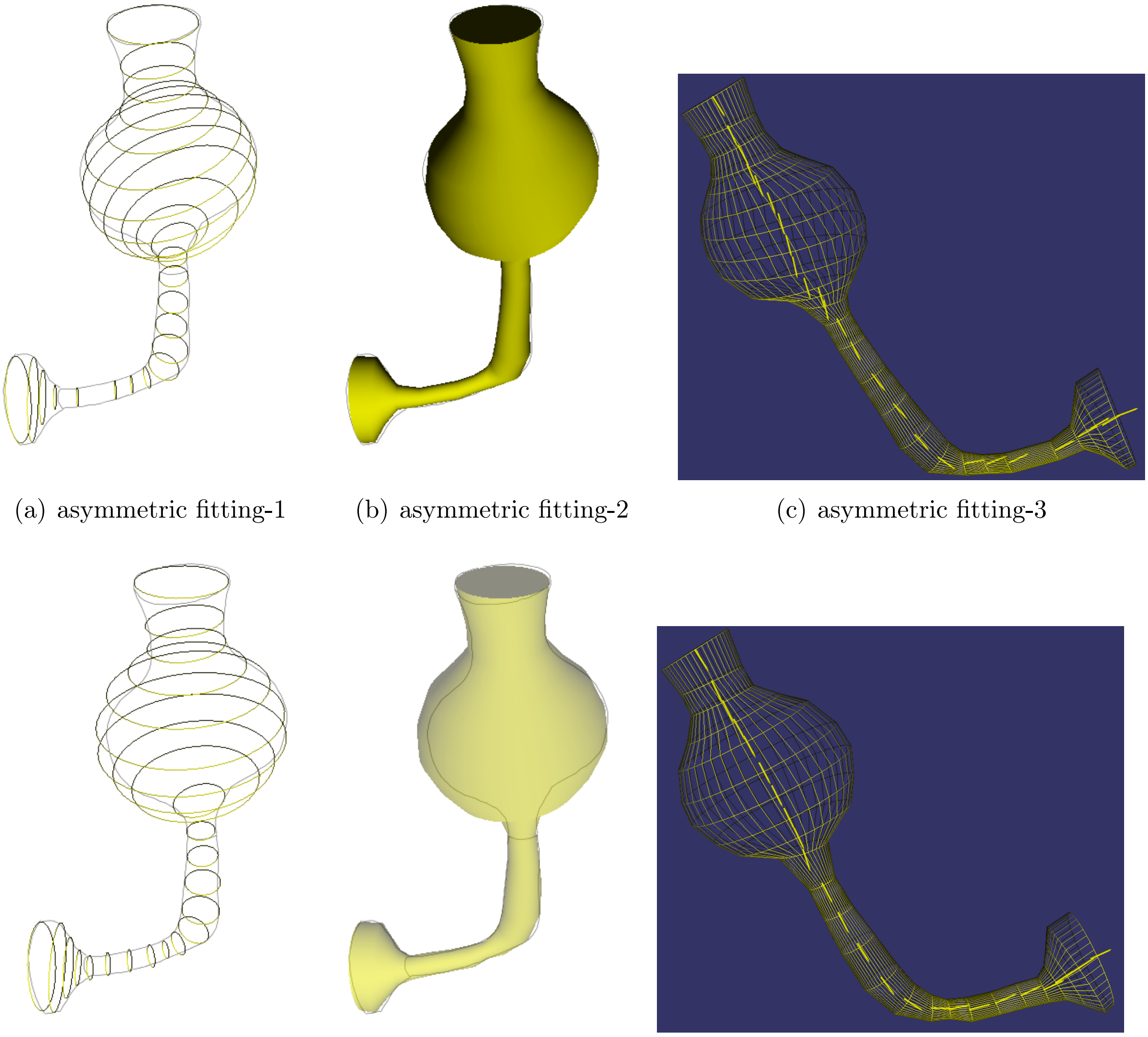
(c) model rendering-2

(d) model rendering-3

(e) model rendered from a different point of view

**Figure 7.25:** A 3D model of a bended pipe with different renderings: the image at (c) is the gradient image.

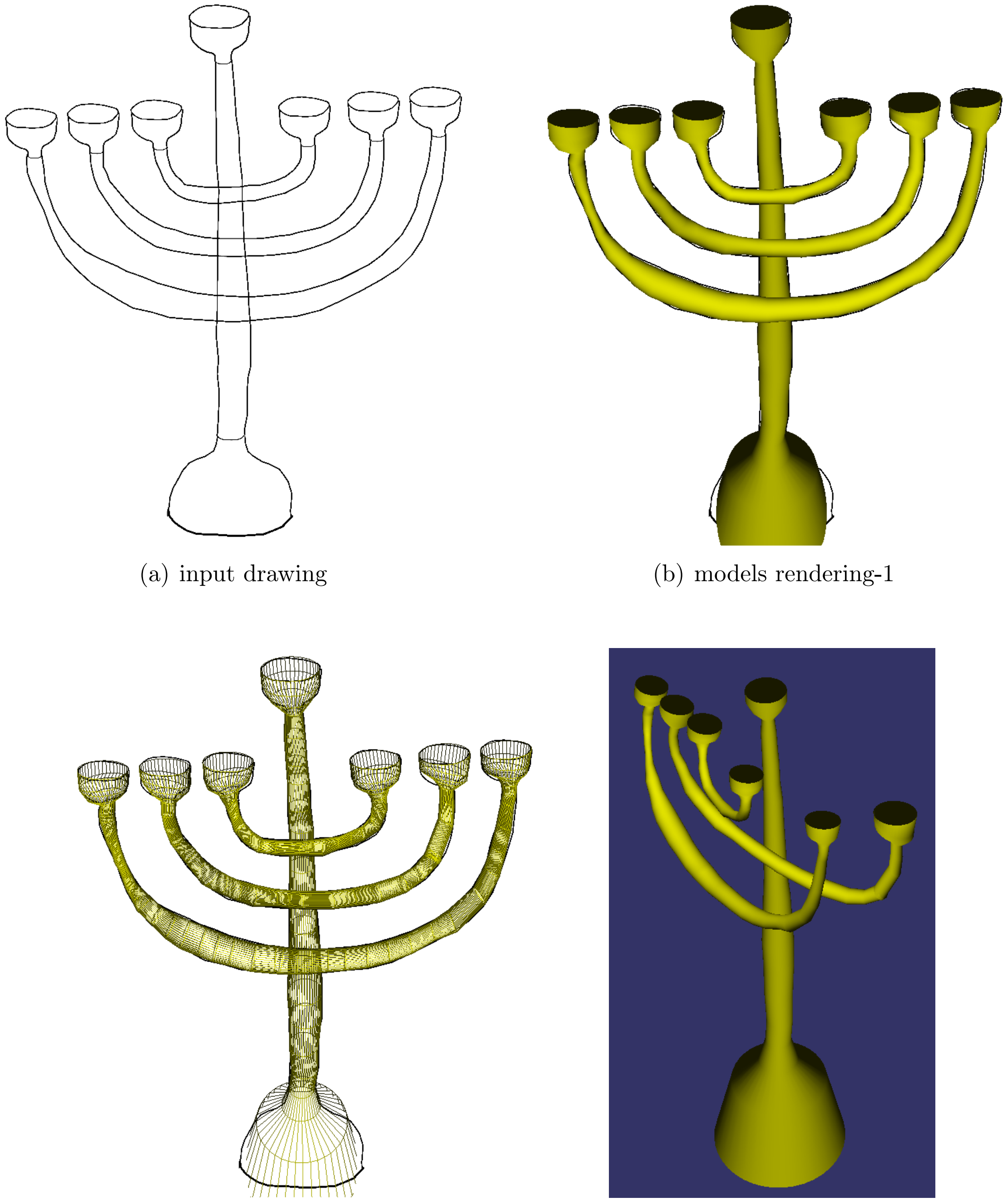
*Figures* 7.26(d) and 7.27 display two reconstructed models from drawings. The first drawing is a wall lamp composed of a single generalized cylinder whereas the second drawing is a menorah composed of four generalized cylinders one of which is straight.


(a) asymmetric fitting-1 (b) asymmetric fitting-2 (c) asymmetric fitting-3

(d) symmetric fitting-1 (e) symmetric fitting-2 (f) symmetric fitting-3

**Figure 7.26:** A 3D model of a wall lamp reconstructed from a drawing with two different major axis fitting strategy: symmetric (a, b, c) and asymmetric (d, e, f). (a, d) display the projections of the reconstructed 3D circles, (b, e) display the corresponding models and (c, f) display the models with a different viewing point.

Recall that users approximate the axis of the generalized cylinder by a piecewise 2D curve on the input image. At the endpoints of the linear segments (constituting the piecewise curve) the previously fit major axis is copied to the current point and it is fit to the outline of the generalized cylinder. This fitting is executed by first rotating the major axis w.r.t the bend of the axis and then shooting inward and outward rays form the major axis endpoints. The fitting is completed by updating the positions of the major axis endpoints according to the ray shooting results. Notice that, if the major axis is not scaled homogeneously, the centre (midpoint) of the major axis deviates from the user clicked point. This fitting strategy is called asymmetric major axis fitting. Another strategy called symmetric major axis fitting forces homogeneous scaling of the major axes which keeps the location of the major axis centres at the user clicked points. However, if the projection of the model in the input image is not symmetric, then the reconstructed model's projection does not fit to the outline of the object (*Figure* 7.26(e)).



(a) input drawing

(b) models rendering-1

(c) models rendering-2

(d) models rendered from a different point of view

**Figure 7.27:** A 3D model of a menorah reconstructed from a drawing: note that each generalized cylinder is reconstructed independently, object modelling by merging the separately constructed parts is not supported by the current version of or modelling system.

The generalized cylinders in the *Figure* 7.27 are modelled separately and when they are rendered with a different point of view, it can be seen that their locations are not coherent with each other to form the model of the menorah object. Part based object modelling remains as a future work.

Observe the projection fit error at the end of the straight generalized cylinder in *Figures* 7.27(b) and 7.27(c). As the angles between the viewing rays (from camera centre to circle centres) and the circle normals decrease, the assumption of a major axis (corresponding to the perspective projection of a 3D circle whose normal is approximately parallel to the axis of the generalized cylinder) whose endpoints are on the silhouette of the straight cylinder becomes invalid. Therefore, the projection fit error is relatively high under these circumstances.

To conclude, in our system we have three different prior constraints one of which can be selected by the user before starting to modelling. The first prior is the constant depth prior which leads to oblique generalized cylinders. The second prior is the planar axis prior which requires drawing the projection of two circles at both ends of the generalized cylinders. The normals of the reconstructed 3D circles are then used to compute the axis plane which is not possible when the two circle normals are parallel or collinear. And the third prior is used to model straight generalized cylinders for which all the circle centres lie on a 3D line segment. This prior suffers from the inaccurate major axis estimations under the circumstances described in the previous paragraph.

## 7.3 Implementation Details

Our modelling system is programmed in C++ using the libraries: OpenSceneGraph[9] (based on OpenGL) for 3D graphics, Orfeo Toolbox[10] for image processing and WxWidgets[11] for GUI development.

In OpenGL, a camera projection matrix performs two things cascaded to each other: projects the 3D points (in the camera coordinate frame) onto the near clipping plane (i) and then computes the clipping coordinates for the projected points (ii). The first part is related to the camera model and the second part is related to the elimination of the projected points that were originally outside the viewing volume (frustum). One can decompose the camera projection matrix into two, change the camera model and recompose the camera projection matrix again. As a result, a camera model other than the default pinhole model can be implemented within OpenGL. However, for other camera models all the derivations given in *Section* 7.2.4 have to be re-done.

In OpenGL, the viewing volume of a camera is defined by the six clipping planes: left, right, bottom, top, near and far. If the viewing volume is symmetric (in $x$ and $y$ axes along the line of sight) then it can be defined by four parameters: vertical field of view, aspect ratio, near and far. We defined our camera model with a symmetric viewing volume. The aspect ratio is computed from the input image size which is fit to the viewport on the screen (aspect ratio = viewport width/viewport height, therefore the image is displayed without distortion). The vertical field of view is set to a typical value of 45 degrees for a standard application and the near and far values are selected as 1.0 and 100.0 respectively.

In OpenGL, a 3D point in the camera coordinate frame passes through a series of transformations until it is mapped to a 2D point on the screen. First, it is multiplied by the camera projection matrix which transforms the point to the clipping space (first into projection

[9] http://www.openscenegraph.org/

[10] https://www.orfeo-toolbox.org/

[11] https://www.wxwidgets.org/

space and then into clipping space) in which the clipping (eliminating the non-visible points) and perspective division (converting homogeneous to Cartesian representation) is performed. After perspective division, normalized device coordinates are obtained which are then converted to the window or screen coordinates by view-port transformation. The equations derived in *Section* 7.2.4 start from the projected space but in a real system, the user clicked points are in the window coordinates and they have to be transformed to the projection space.

## 7.4 Conclusion

In this chapter, we presented a system for modelling generalized cylinders from a single input image. The problem of geometric reconstruction from a single image is ill-posed since the depth information is lost during the imaging process (perspective projection). However, with the help of some prior constraints approximate solutions (plausible models) can be obtained.

As a contribution, (i) analytical computation of 3D circles from their perspective and orthographic projections are explained in detail including the special and degenerate cases; (ii) a closed form solution to the centre of a 3D circle is found when its normal is known and its perspective projection is partially known (only the major axis); (iii) this closed form solution is used to reconstruct (estimate) the geometry of the generalized cylinders from single images together with some user selected prior constraints related to the axis of the generalized cylinders.

Chen et al. [2013] developed a more complete system for modelling part-based objects which depends on an optimization problem for finding the positions of the separately modelled parts as well as for compensating the ratio preserving projection (in order to minimize the perspective projection fit error) which is used to model individual parts. Our system does not support the modelling of objects that are composed of several parts. On the other hand, we used perspective projection in our computations and the resultant models are circular and homogeneous unlike the their system which may warp the 3D circles into 3D ellipses for the minimization of the perspective projection fit error.

The presented system is in a preliminary state and there is plenty of room for further improvement most of which have already been discussed. Here are some future works: (i) texturing from the input images, (ii) modelling primitives other than generalized cylinders, (iii) modelling objects that are composed of multiple primitive types, (iv) integrating better edge detection methods to improve major axis fitting, (v) integrating a continuous sweep for the axis approximation (i.e. sampling projection of the axis with a greater number of points), (vi) searching for other prior constraints especially for fixing the radii of the reconstructed circles, (vii) integrating the other views of the modelled objects into the modelling procedure so that the reconstructed models can be truly scaled and geo-referenced as well as the quality of the reconstructed models can be improved.

# Chapter 8

# Conclusion

The work that has been done in the main part of this thesis can be considered as an effort to design new ways of representing/storing geo-data such that it enables easy GIS application development. As the time passes, more and more amount of data is being produced with the spread of advancing technology and emerging new sensors. Parallel to the data, there will be more applications flourishing which will bring the necessity of better ways of data management and organization (dealing with big data). The GIS application development pipeline presented in this thesis is composed of two main steps: (i) object-based hierarchical modelling of the interested region in 2D (ii) developing the application specific algorithms/pipelines and applying them on the generated 2D models in the first step. These 2D models are generic in the sense that they can be used by different algorithms for different applications. The next paragraph identifies a potential usage of such a generic model.

Traditionally, geo-data related to different thematic concepts are stored separately without any coherence in between. An example to such data can be the cadastral maps and the road network graphs. Therefore, once the physical environments change, all of the related data have to be updated independently. 2D models capturing the environments in many aspects similar to ours can be utilized to merge all kinds of different purposed databases at the very basic level. Then, when a modification is done on this model, all other upper layers of data can be computed from the underlying 2D model. That is, each of these upper layer data can be considered as the output of a separate application processing the base 2D model. The pedestrian network graphs computed in this thesis can also be considered as an example to such an application.

As the amount of semantic data stored in the base models increase there will be new application ideas that can emerge by the analysis or combination of different aspects of the data. Especially, the spatial statistical data collected in time might be very useful to identify patterns and relationships between the geospatial objects and the entities interacting with them. Integrating multi-diverse data and making decisions based on these might be one of the key components of the future intelligent agents. In this regard, the following paragraphs are presenting some ideas (inspired by the paper Goodchild and Longley [1999]) related to feature of GIS which will need generic, efficient and effective ways of handling (i.e. organization, management and usage) data.

Real time locating systems using the satellite positioning systems (e.g. GPS, GLONASS, Galileo) and ground based systems (e.g. mobile cellular networks, indoor positioning systems based on RFID or Wifi) are being used all around us. Besides, the number of sensors receiving signals from such systems is getting bigger and bigger as more and more people are possessing the portable/wearable smart devices. In addition, location based services have already become popular for navigation or for entertainment in social networks. In the near feature, the emergence of more smart applications based on locations of people, vehicles, pets, etc. might be on the horizon. Examples of such applications can be: collecting data on the movement of people and designing better public spaces based on the calculated statistics, reaching quickly to the people that are in emergency conditions, finding lost pets, etc. Although such applications are welcome for many people, some may have privacy concerns which needs to be carefully assigned to application designs and standards to comply with the ethical issues.

OpenStreetMap[1] is a very successful example of a crowd sourcing project. People from all around the world are collaborating with each other for generating a worldwide map. The advantage of such a map is that anyone can use his/her local knowledge to contribute. In the near future, there will be more crowd source applications. For instance, point clouds generated by portable mobile laser scanners can be registered/merged to generate a worldwide 3D point cloud and automatic/semi-automatic 3D modelling can be done on this data. Researches might need to work on better and easier platforms (e.g. new types of user interfaces, standards and protocols) for encouraging people to collaborate on such crowd source projects. Additionally, new algorithms and application pipelines will be needed in order to make the best out of the collectively collected data.

Although there exist 3D models being used in GISs, many of the current systems are still based on 1D and 2D. In the future GISs, higher dimensions (3D, 4D → 3D + time, etc.) will be dominating and there will be more cross-disciplinary applications involving many fields of science and technology such as computer graphics, computer vision and remote sensing. A virtual reality GIS can be a good candidate for such an application. As a result of evolving branches of science, the growing space between these branches will be filled with more and more multi-disciplinary research. The last chapter of the thesis can be considered as a research carried out in this direction.

[1] https://www.openstreetmap.org

# Appendices

# Appendix A

# Geometric Data Structures

The advantages and disadvantages of some of the geometric data structures are discussed in *Section* 2.2. In this appendix, simple definitions of some geometric data structures are given in pseudo C++ code without actual implementations. This appendix is complementary to the *section* 2.2 for the interested reader.

## A.1 Polygon Soup

```
// vertex definition
struct Vertex {
    double x, y;
};

// face definition
struct Face {
    std::vector<Vertex> vertices;
    FaceDataObject data;
};

// container for faces
std::vector<Face> faces;
```

**Listing A.1:** An example definition of the polygon soup (spaghetti) data structure

## A.2 Shared Vertex

```
// vertex definition
    struct Vertex {
        double x, y;
        VertexDataObject      data;
    };
```

```
// face definition
struct Face {
    std::vector<VertexRef> vertices;
    FaceDataObject data;
};

// container for vertices and faces
std::vector<Vertex>      vertices;
std::vector<Face> faces;
```

**Listing A.2:** An example definition of the shared vertex data structure

## A.3 Winged-Edge

```
// edge definition
    struct Edge {
        VertexRef vertex_start;
        VertexRef vertex_end;
        FaceRef left_face;
        FaceRef right_face;
        EdgeRef left_successor;
        EdgeRef left_predecessor;
        EdgeRef right_successor;
        EdgeRef right_predecessor;
        EdgeDataObject data;
    };

    // vertex definition
    struct Vertex {
        double x, y;
        EdgeRef edge;
        VertexDataObject data;
    };

    // face definition
    struct Face {
        EdgeRef edge;
        FaceDataObject data;
    };

    // containers
    std::vector<Vertex> vertices;
    std::vector<Edge> edges;
```

```
std::vector<Face> faces;
```

**Listing A.3:** An example definition of the Winged-edge data structure

## A.4 Half-Edge

```
// half-edge definition
struct Halfedge {
    HalfedgeRef next_halfedge;
    HalfedgeRef opposite_halfedge;
    FaceRef face;
    VertexRef to_vertex;
};

// vertex definition
struct Vertex {
    double x, y;
    HalfedgeRef outgoing_halfedge;
    FaceDataObject data;
};

// face definition
struct Face {
    HalfedgeRef halfedge;
    FaceDataObject data;
};

// containers
std::vector<Vertex> vertices;
std::vector<Halfedge> halfedges;
std::vector<Face> faces;
```

**Listing A.4:** An example definition of the Half-edge data structure

# Appendix B

# Delaunay Triangulation Extended Notes

This appendix is composed of two parts. In the first part, a proof for the legal and illegal edges of a Delaunay triangulation is given. The proof is quite simple but it is omitted in the modern text books which motivated the author to put it into the appendix. The second part is devoted to the explanation of the merge step for the divide and conquer Delaunay triangulation algorithm. Each step of the merge algorithm is displayed with an illustrative figure, which eases to understand the algorithm [Lee and Schachter, 1980].

## B.1 Delaunay Triangulation Legal and Illegal edges

Please refer to *Section* 4.1.2 for the definition of legal and illegal edges within a Delaunay triangulation. In this section of the appendix, a proof for the definition of legal and illegal edges is given using the very basic geometry known more than two thousands years.

*Euclid Elements - Book I - Proposition-32:* In any triangle, if one of the sides is produced, then the exterior angle equals the sum of the two interior and opposite angles, and the sum of the three interior angles of the triangle equals two right angles (*Figure* B.1).

*Euclid Elements - Book III - Proposition-20:* In a circle the angle at the center is double the angle at the circumference when the angles have the same circumference as base (*Figure* B.1).

*Euclid Elements - Book III - Proposition-21:* In a circle the angles in the same segment equal one another (*Figure* B.1).

*Proposition-B1:* Let $A, B, C$ be three non-collinear points in Euclidean plane ($R^2$). Let also $D, E \in R^2$ be two other distinct points such that $D$ is on the bounded side and $E$ is on the unbounded side of the circumcircle of $\triangle ABC$. Then the angles formed by these points has the order given in *Equation* 1 (*Figure* B.2).

$$\angle(BDC) \quad > \quad \angle(BAC) \quad > \quad \angle(BEC) \tag{1}$$

*Proposition-B1* can easily be proved using Euclid's elements, Book-I Proposition-32 and Book-III Proposition-20 and Proposition-21. The *Figures* B.1 and B.2 are sufficiently

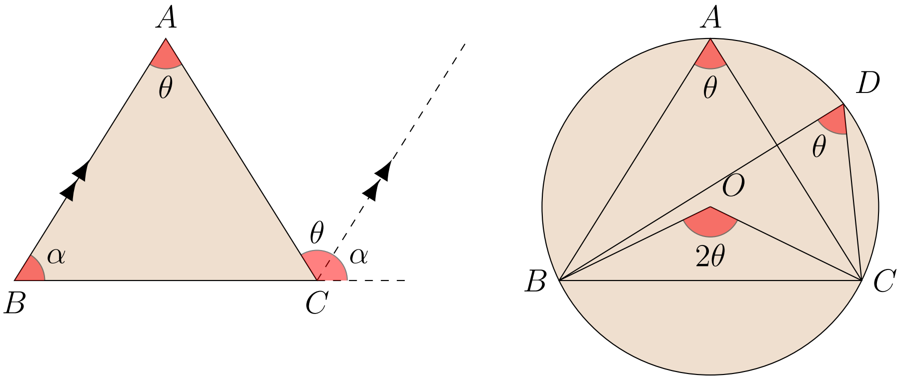


**Figure B.1:** Euclid's Elements: Book-I Proposition-32 (left) and Book-III Proposition-20 and Proposition-21 (right)

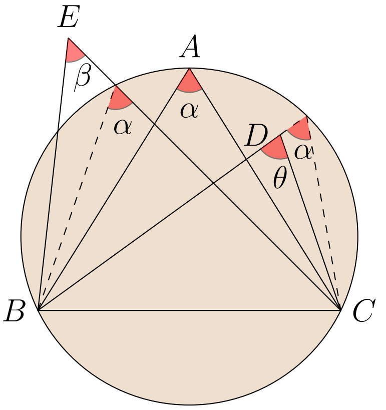


**Figure B.2:** Proposition-B1: $\theta > \alpha > \beta$

clear that no more explanation is needed.

*Proposition-B2:* Let $\overline{AC}$ be an internal edge of a triangulation between the triangles $\triangle ABC$ and $\triangle ACD$. Then, $\overline{AC}$ is a legal edge if $D$ is outside the circumcircle of $\triangle ABC$ and an illegal edge if $D$ is inside the circumcircle.

*Proof:* Let $D$ be is inside the circumcircle of $\triangle ABC$. We'll prove that the edge $\overline{AC}$ is illegal. *Figure* B.3 displays the angles constructed by the two diagonals of the quadrangle $\Diamond ABCD$.

Let $V_{AC} = \{\theta_1, \theta_2, \theta_3, \theta_4, \alpha_1+\alpha_2, \alpha_3+\alpha_4\}$ and $V_{BD} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \theta_1+\theta_4, \theta_2+\theta_3\}$ be two sets representing the angles formed by the diagonals $\overline{AC}$ and $\overline{BD}$ respectively. Observe in *Figure* B.3 that $\alpha_i > \theta_i$ for $i \in \{1, 2, 3, 4\}$ by Proposition-B1. That is, for any element of the set $V_BD$, there exists a smaller element in $V_AC$ providing the edge $\overline{AC}$ is illegal.

The other part of the algorithm, i.e. legality of edge $\overline{BD}$ can be proven similarly way.

## B.2 Delaunay Triangulation Merge Algorithm

Lee and Schachter [1980] developed the first divide and conquer algorithm for generating Delaunay triangulations. The backbone of their algorithm is the merging of two separate Delaunay triangulations. Let $S_L$ and $S_R$ be two point sets which are separated by a

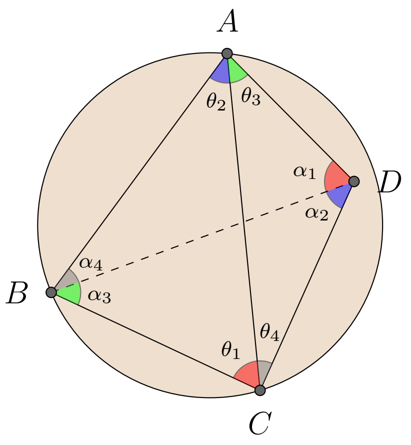


**Figure B.3:** Angles constructed by two diagonals

vertical line $l$ where $S_L$ lies on the left and $S_R$ lies on the right of $l$. In addition, let the corresponding Delaunay triangulations be $DT(S_L)$ and $DT(S_R)$ (*Figure* B.4). Finally, let $S = S_L \cup S_R$ be the joint point set and $DT(S)$ denotes its Delaunay triangulation.

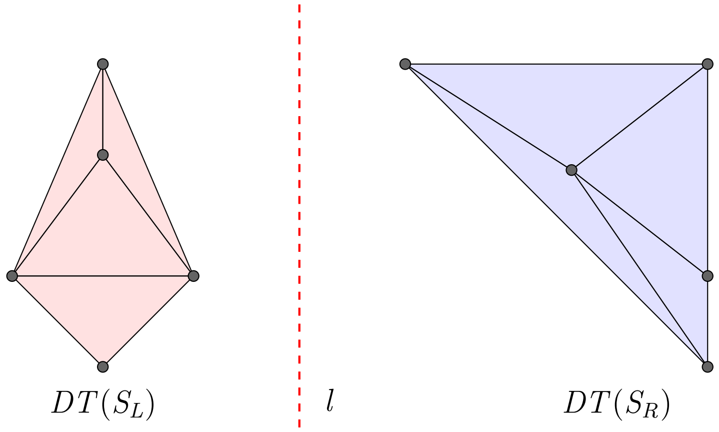


**Figure B.4:** Delaunay triangulation of two point sets: $DT(S_L)$ (on the left) and $DT(S_R)$ (on the right)

Let's call the edges of $DT(S_L)$ and $DT(S_R)$ as $LL$-edges and $RR$-edges, respectively. At the end of the merging process $DT(S)$ will be composed of $LL$-edges and $RR - edges$ besides the newly generated $LR$-edges. $LR$-edges connect two points $p_L, p_R$ where $p_L \in S_L$ and $p_R \in S_R$. During the merging process, some of the $LL$-edges and $RR$-edges might become illegal. Those become illegal are deleted from $DT(S)$. Please refer to *Section* 4.1 in order to recall the definition of legal and illegal edges in a Delaunay triangulation.

The algorithm starts by finding the Lower Common Tangent (LCT) between $DT(S_L)$ and $DT(S_R)$. This sub-problem is well known from the merging of two convex hulls [Preparata and Hong, 1977]. LCT is the first $LR$-edge that is inserted into $DT(S)$. After the insertion of the first $LR$-edge, we search for the next $LR$-edges to be inserted into $DT(S)$. Each consecutive $LR$-edge will be located above the previous one. Moreover, a pair of consecutive $LR$-edges share an end-point either at the left or right side. Starting from LCT, next $LR$-edges are found iteratively until the last $LR$-edge is reached. The last $LR$-edge will be the Upper Common Tangent (UCT) of $DT(S_L)$ and $DT(S_R)$.

Let's call the current or recently added $LR$-edge as base $LR$-edge and let's denote the end-points of a base $LR$-edge with $L$ and $R$ such that $L \in S_L$ and $R \in S_R$. Either $L$ or $R$ will be used as one of the end-point of the next $LR$-edge. For instance, if $L$ is used, then the other end-point of the next $LR$-edge will be selected from the right side and vice versa.

For the end-point selection let's define two sets one for the left and one for the right side, containing the candidate points. Let's denote the candidate point sets as $CS_L = \{C_{L1}, C_{L2}, ..., C_{Ln}\}$ and $CS_R = \{C_{R1}, C_{R2}, ..., C_{Rm}\}$.

Any point $C_{Li} \in CS_L$ must comply to the following criteria.

1. $C_{Li}$ must be connected to $L$ by an $LL$-edge.
2. The point triple $(R, L, C_{Li})$ must make a right turn.
3. The angle formed by $(R, L, C_{Li})$ must be smaller than the angle $(R, L, C_{L(i+1)})$.

Similarly, any point $C_{Ri} \in CS_R$ must comply to the following criteria.

1. $C_{Ri}$ must be connected to $R$ by an $RR$-edge.
2. The point triple $(L, R, C_{Ri})$ must make a left turn.
3. The angle formed by $(L, R, C_{Ri})$ must be smaller than the angle $(L, R, C_{R(i+1)})$.

Observe *Figure* B.5 for the constructed candidate point sets.

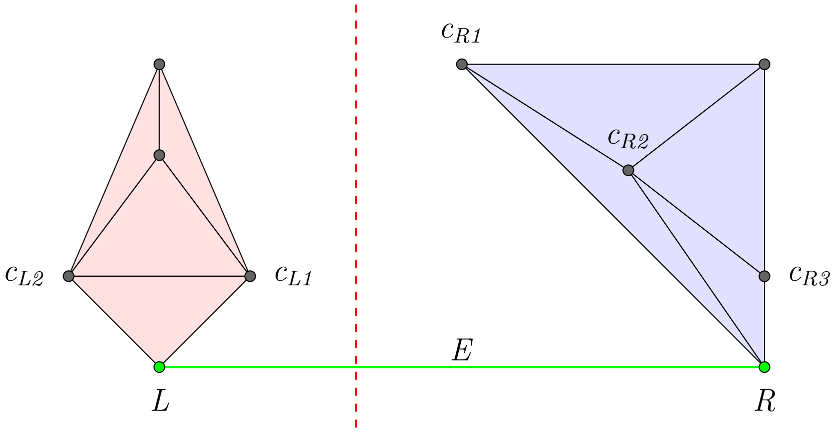


**Figure B.5:** The first edge added to $DT(S)$ is the lower common tangent of $DT(S_L)$ and $DT(S_R)$ labelled as $E$. $E$ is the recently added $LR$-edge, i.e, it is the base $LR$-edge. Its end-points are marked with $L$ and $R$. The points satisfying the candidate point criteria are also labelled: $CS_L = \{C_{L1}, C_{L2}\}$ and $CS_R = \{C_{R1}, C_{R2}, C_{R3}\}$.

After constructing the candidate point sets we inspect each candidate point one by one starting from the beginning. The first candidate point satisfying the one final criterion will be the nominated for the final candidate point representing its set ($CS_L$ or $CS_R$). The final criterion is: the circumcircle defined by $L$, $R$ and the current candidate point must not contain the next potential candidate in its interior. This criterion is also called as

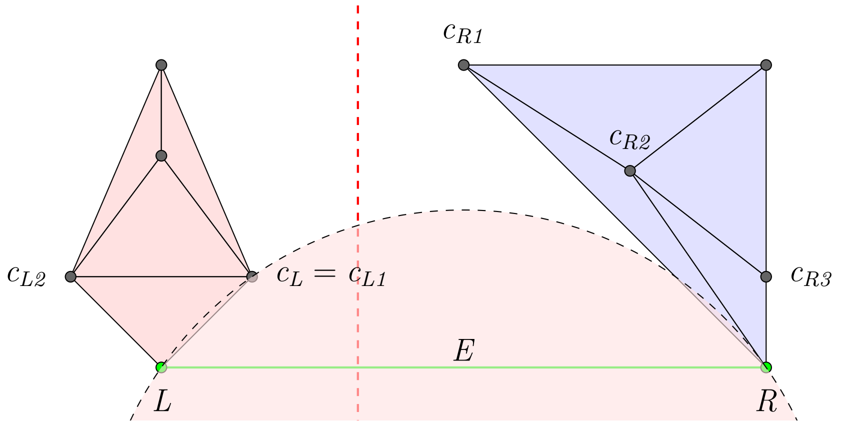


**Figure B.6:** $C_L = C_{L1}$ passes the circumcircle test and becomes the nominee of $CS_L$ for the final candidate.

circumcircle test. Observe the *Figure* B.6, the first candidate point passes the circumcircle test and it's selected as a nominee ($C_L$) for the final candidate from the left side.

On the right side, the first candidate point fails the circumcircle test. When a candidate point fails the test, it makes the edge connecting the candidate point to end-point of the base $LR$-edge on the same side becomes illegal and it has to be deleted (*Figure* B.7).

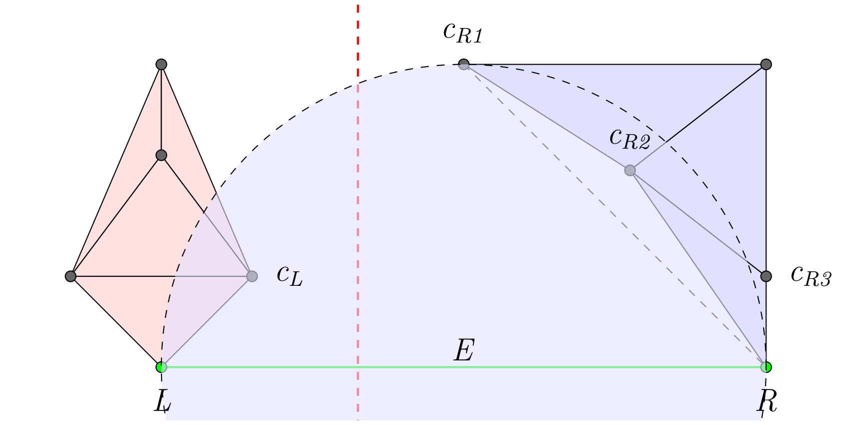


**Figure B.7:** $C_{R1}$ fails the circumcircle test and the corresponding $RR$-edge is deleted from $DT(S)$. Observe that the next potential candidate $C_{R2}$ is in the interior of the circumcircle.

After the failure of the first candidate point we move onto the second candidate point. This time, the circumcircle test is passed and the nominee ($C_R$) for the final candidate point from the right side is selected as $C_{R2}$ (*Figure* B.8).

At this point, we have two nominees for the final candidate point one from left: $C_L \in S_L$ and one from right: $C_R \in S_R$. We need to select one of them and the test for this selection is one more time, the circumcircle test. If $C_R$ is not contained in the interior of the circle defined by $(L, R, C_L)$, then $C_L$ defines the next $LR$-edge and vice-versa. By the guaranteed existence of the Delaunay triangulation, at least one of the candidates will satisfy this and by the uniqueness of the Delaunay triangulation, only one candidate will satisfy this except in the case when the four points are co-circular (both candidates pass the circumcircle test and one of them can be chosen). For our example, $C_L$ is selected as the end-point of the next $LR$-edge. Hence, automatically $R$ becomes the other end of the

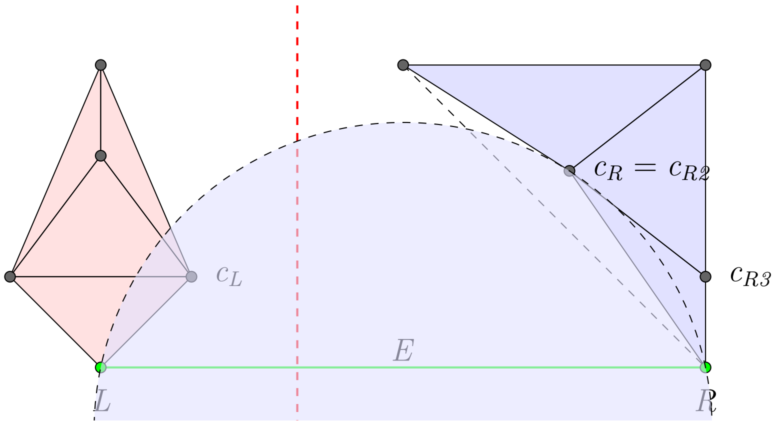


**Figure B.8:** $C_R = C_{R2}$ passes the circumcircle test and becomes the nominee of $CS_R$ for the final candidate.

next $LR$-edge (*Figure* B.9 and *Figure* B.10).

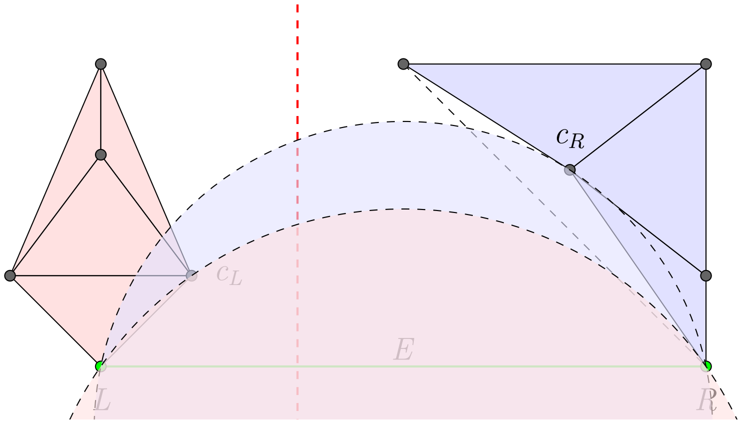


**Figure B.9:** Circumcirle test for the selection of the end point of the next $LR$-edge. The interior of the $(L, R, C_L)$ circumcircle does not contain $C_R$ whereas the interior of $(L, R, C_R)$ circumcirle contains $C_L$. Hence $C_L$ is selected as the end-point of the next $LR$-edge.

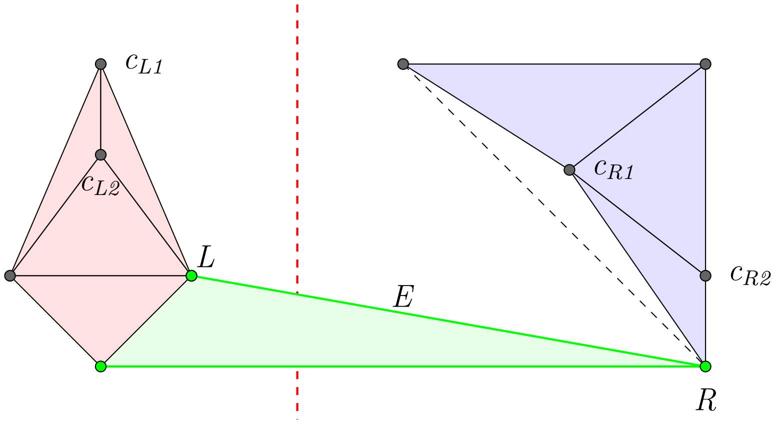


**Figure B.10:** The next $LR$-edge has been added to $DT(S)$, which becomes the base $LR$-edge. As a result a new triangle is constructed within $DT(S)$. The new candidate point sets are then generated as: $CS_L = \{C_{L1}, C_{L2}\}$ and $CS_R = \{C_{R1}, C_{R2}\}$.

While applying the process iteratively, we might not have a nominee for the final candidate point from left or right side. This might happen if the corresponding candidate point set is empty or none of the points in the candidate point set passes the circumcircle test. In this case, the nominee from the other side is automatically selected as the final candidate point. Finally, we end the process when we do not have any nominees both from left and right sides. In fact, such a case occurs only after we reached the UCT of $DT(S_L)$ and $DT(S_R)$.

For the example point set that we used to discuss the Delaunay triangulation merge algorithm, $S = S_L \cup S_R$, *Figure* B.11 displays the final output $DT(S)$ and *Figure* B.12 displays the circumcircles of the merging triangles.

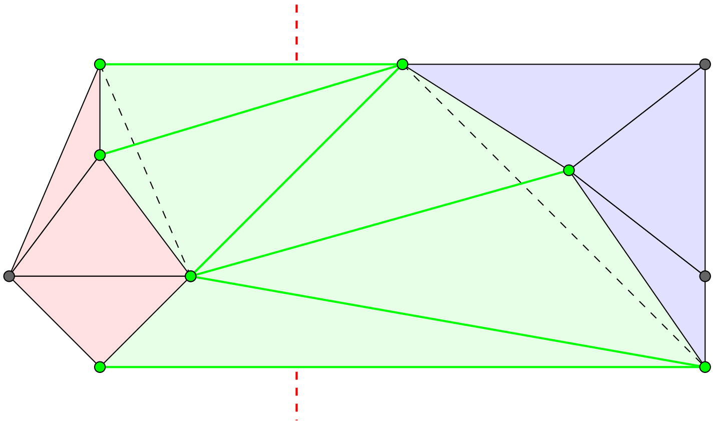


**Figure B.11:** Merged Delaunay triangulation $DT(S)$: Red and blue triangles illustrate the triangles within $DT(S)$ which are preserved from $DT(S_L)$ and $DT(S_R)$ respectively. Furthermore, the green triangles are generated in order to merge $DT(S_L)$ and $DT(S_R)$

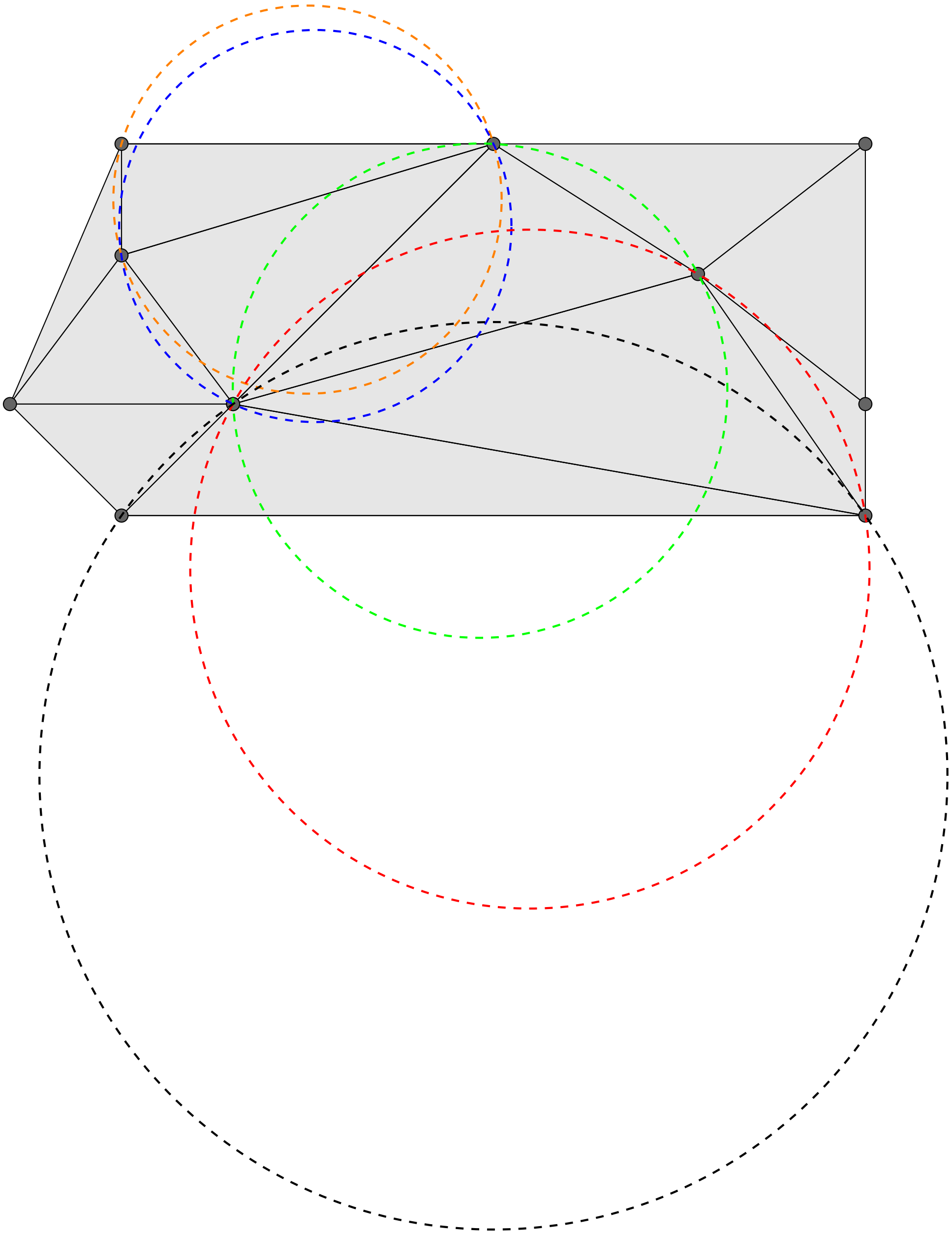


Figure B.12: Circumcircles of the merging triangles

# Bibliography

P. K. Agarwal and M. Sharir. Arrangements and their applications. In *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, 1998. 30

O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. In *Computing and Combinatorics*, volume 1090 of *Lecture Notes in Computer Science*, pages 117–126. Springer Berlin Heidelberg, 1996. 70

O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gärtner. A novel type of skeleton for polygons. In *J.UCS The Journal of Universal Computer Science*, pages 752–761. Springer Berlin Heidelberg, 1996. 67, 69

O. Aichholzer, F. Hurtado, and M. Noy. A lower bound on the number of triangulations of planar point sets. *Computational Geometry*, 29(2):135 – 145, 2004. 49, 70

H. Alt, O. Cheong, and A. Vigneron. The voronoi diagram of curved objects. *Discrete and Computational Geometry*, 34(3):439–453, 2005. 56

D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. Lafon, R. Lyon, A Ogale, L. Vincent, and J. Weaver. Google street view: capturing the world at street level. *Computer*, 43(6): 32–38, 2010. 117

ANSI/IEEE Standard 754. American National Standards Institute and Institute of Electrical and Electronic Engineers. IEEE Standard for Binary Floating-Point Arithmetic , 1985. 4, 83

H. Anton and C. Rorres. *Elementary Linear Algebra.* Wiley, 9th edition, 2005. Applications Version, See section 9.7 for Quadratic Surfaces. 125

P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 33(5):898–916, 2011. 121

O. Arikan, S. Chenney, and D. A. Forsyth. Efficient multi-agent path planning. In *In Proceedings of the 2001 Eurographics Workshop on Animation and Simulation*, pages 151–162, 2001. 94

D. Attali, J. D. Boissonnat, and H. Edelsbrunner. Stability and computation of medial axes - a state-of-the-art report. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, Mathematics and Visualization, pages 109–125. Springer Berlin Heidelberg, 2009. 71

F. Aurenhammer and R. Klein. Voronoi diagrams. In *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers B.V. North-Holland, 2000. 53, 55, 65

D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1–3):21 – 46, 1996. 49

M. G. Ballester, M. R. Pérez, and J. Stuiver. Automatic pedestrian network generation. In *Proceedings of 14th AGILE International Conference on GIS*, pages 1–13, 2011. 92

G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall PTR, 1998. 24

B. G. Baumgart. A polyhedron representation for computer vision. In *AFIPS '75 Proceedings of the May 19-22, National Computer Conference and Exposition*, pages 589–596. ACM, 1975. 20

L. Beale, K. Field, D. Briggs, P. Picton, and H. Matthews. Mapping for wheelchair users: Route navigation in urban spaces. *The Cartographic Journal*, 43:68–81, 2006. 92, 93

E. Berberich, E. Fogel, D. Halperin, M. Kerber, and O. Setter. Arrangements on parametric surfaces ii: Concretizations and applications. *Mathematics in Computer Science*, 4 (1):67–91, 2010a. 30

E. Berberich, E. Fogel, D. Halperin, K. Mehlhorn, and R. Wein. Arrangements on parametric surfaces i: General framework and infrastructure. *Mathematics in Computer Science*, 4(1):45–66, 2010b. 30

J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps.* Paris, Mouton / Gauthier-Villars, 1967. 45

P. Bhattacharya and M. L. Gavrilova. Roadmap-based path planning - using the voronoi diagram for a clearance-based shortest path. *Robotics Automation Magazine, IEEE*, 15 (2):58–66, 2008. 94

T. O. Binford. Visual perception by computer. In *IEEE Conference on Systems and Controls*, 1971. 119, 121

H. Blum. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, 1967. 64, 119

M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. OpenMesh - a generic and efficient polygon mesh data structure. In *Proceedings of OpenSG Symposium*, 2002. 21

M. Botsch, M. Pauly, L. Kobbelt, P. Alliez, B. Lévy, S. Bischoff, and C. Rössl. Geometric modeling based on polygonal meshes. In *ACM SIGGRAPH 2007 Courses*, 2007. 21

J. M. Boyer. On the cutting edge: Simplified o(n) planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004. 25

M. Brédif, B. Vallet, and B. Ferrand. Distributed dimensionality-based rendering of lidar point clouds. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XL-3/W3*, 2015. 11, 108

O. Byer, F. Lazebnik, and D. L. Smeltzer. *Methods for Euclidean Geometry.* MAA, 2010. page 124, Theorem 6.3. 112

S. Campagna, L. Kobbelt, and H. P. Seidel. Directed edges - a scalable representation for triangle meshes. *ACM Journal of Graphics Tools*, 3(4):1–12, 1998. 21

R. J. Campbell and P. J. Flynn. A survey of free-form object representation and recognition techniques. *Computer Vision and Image Understanding*, 81(2):166–210, 2001. 117

V. Carette, Mostafavi M. A., R. Devillers, G. Rose, and L. H. Beni. Extending marine gis capabilities: 3D representation of fish aggregations using delaunay tetrahedralisation and alpha shapes. *Geomatica*, 62(4):361–374, 2008. 63

T. Chen, Z. Zhu, A. Shamir, S. Hu, and D. Cohen-Or. 3-sweep: Extracting editable objects from a single photo. *ACM Transactions on Graphics (TOG)*, 32(6):195, 2013. 12, 117, 119, 120, 121, 136, 137, 141, 142, 152

M. Cheng. Curve structure extraction for cartoon images. In *Proceedings of the 5th Joint Conference on Harmonious Human Machine Environment*, pages 13–25, 2009. 121

S. W. Cheng and A. Vigneron. Motorcycle graphs and straight skeletons. *Algorithmica*, 47(2):159–182, 2007. 70

S. W. Cheng, L. Mencel, and A. Vigneron. A faster algorithm for computing straight skeletons. In *Algorithms - ESA*, volume 8737 of *Lecture Notes in Computer Science*, pages 272–283. Springer Berlin Heidelberg, 2014. 70

F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete & Computational Geometry*, 21(3):405–420, 1999. 65

G. K. W. Chin, K. P. Van Niel, B. Giles-Corti, and M. Knuiman. Accessibility and connectivity in physical activity studies: The impact of missing pedestrian data. *American Journal of Preventive Medicine*, 46(1):41–45, 2008. 91, 92

J. K. Clifton, A. D. L. Smith, and D. Rodriguez. The development and testing of an audit for the pedestrian environment. *Landscape and Urban Planning*, 80(1–2):95 – 110, 2007. 92

C. Colombo, A. Del Bimbo, and F. Pernici. Metric 3d reconstruction and texture acquisition of surfaces of revolution from a single uncalibrated view. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(1):99–114, 2005. 120

T. Culver, J. Keyser, and D. Manocha. Exact computation of the medial axis of a polyhedron. *Computer Aided Geometric Design*, 21(1):65 – 98, 2004. 66

G. Damiand and P. Lienhardt. *Combinatorial Maps - Efficient Data Structures for Computer Graphics and Image Processing.* CRC Press, 2015. 33

M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag Berlin Heidelberg, 1997. 22

J. A. de Loera, J. Rambau, and F. Santos. *Triangulations: Structures for Algorithms and Applications*, volume 25 of *Algorithms and Computation in Mathematics*. Springer-Verlag Berlin Heidelberg, 2010. 47, 49

P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 11–20, 1996. 117

B. Delaunay. Sur la sphere vide. *Bulletin of Academy of Sciences of the USSR*, 7:793–800, 1934. 51

E. D. Demaine, M. L. Demaine, and A. Lubiw. Folding and cutting paper. In *Discrete and Computational Geometry*, volume 1763 of *Lecture Notes in Computer Science*, pages 104–118. Springer Berlin Heidelberg, 2000. 70

S. L. Devadoss and J. O'Rourke. *Discrete and Computational Geometry.* Princeton University Press, 2011. 47, 55

R. A. Dwyer. A faster divide-and-conquer algorithm for constructing delaunay triangulations. *Algorithmica*, 2(1-4):137–151, 1987. 57

J. Döllner, T. H. Kolbe, F. Liecke, T. Sgouros, and K. Teichmann. The virtual 3d city model of berlin-managing, integrating, and communicating complex urban information. In *Proceedings of the 25th Urban Data Management Symposium*, 2006. 119

H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72, 1994. 57, 62, 63

H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *Information Theory, IEEE Transactions on*, 29(4):551–559, 1983. 57

H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15(2):341 – 363, 1986. 30

M. J. Egenhofer and A. U. Frank. Object-oriented modeling for gis. *URISA Journal*, 4 (2):3–19, 1992. 36

B. Elias. Pedestrian navigation - creating a tailored geodatabase for routing. In *Proceedings of Positioning, Navigation and Communication*, pages 41–47, 2007. 92

D. Eppstein and J. Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete & Computational Geometry*, 22(4):569–592, 1999. 69, 70

ESRI. ArcGIS TM : Working with geodatabase topology, an ESRI white paper, 2003. 29

ESRI Shapefile. ESRI (Environmental Systems Research Institute) Shapefile Technical Description, 1998. 23

G. Fangi, F. Fiori, G. Gagliardini, E. Savina, and E. S. Malinverni. Fast and accurate close range 3d modelling by laser scanning system. In *Proceedings of 18th International Symposium of CIPA*, 2001. 117

M. Ferri, F. Mangili, and Viano G. Projective pose estimation of linear and quadratic primitives in monocular computer vision. *CVGIP: Image Understanding*, 58(1):66 – 84, 1993. 124

E. Fogel, O. Setter, and D. Halperin. Exact implementation of arrangements of geodesic arcs on the sphere with applications. *24th European Workshop on Computational Geometry*, pages 83–86, 2008. 30

E. Fogel, Halperin D., and Wein R. *CGAL Arrangements and Their Applications*. Springer-Verlag Berlin Heidelberg, 2012. 32

S. Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1-4):153–174, 1987. 57

L. Fousse, G. Hanrot, V. Lefevre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. Research Report RR-5753, Inria, 2005. 84

J. Françon and Y. Bertrand. Topological 3d-manifolds: a statistical study of the cells. *Theoretical Computer Science*, 234(1–2):233 – 254, 2000. 33

C. Gaisbauer and A. U. Frank. Wayfinding model for pedestrian navigation. In *11th AGILE International Conference on Geographic Information Science*, 2008. 91

R. Geraerts. Planning short paths with clearance using explicit corridors. In *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, pages 1997–2004, 2010. 94

C. M. Gold and M. Dakowicz. The crust and skeleton – applications in gis. In *Proceedings of 2nd International Symposium on Voronoi Diagrams in Science and Engineering*, pages 33–42, 2005. 64, 67

J. Gong and D. Li. Design and implementation of an object-oriented GIS software. In *18th International congress of ISPRS*, volume 31, pages 299–304, 1996. 36

M. O. Gonorov and A. G. Khorev. Object-oriented GIS and representation of multi-detailed data. In *18th International congress of ISPRS*, volume 31, 1996. 36

M. F. Goodchild. Gis and basic research: The national center for gographic information and analysis. *Government Information Quarterly*, 7(3):343–355, 1990. 1

M. F. Goodchild. Geographical information science. 6, 1,. *International Journal of Geographical Information Systems*, 6(1):31–45, 1992. 1

M. F. Goodchild and P. A. Longley. The future of gis and spatial analysis. In *Geographical information systems: principles, techniques, management and applications*, pages 567–580. Wiley, 1999. 153

A. D. Gross and T. E. Boult. An algorithm to recover generalized cylinders from a single intensity view. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, volume 2, pages 790–795, 1990. 120

A. D. Gross and T. E. Boult. Recovery of shgcs from a single intensity view. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(2):161–180, 1996. 120

J. L. Gross, J. Allen, and P. Zhang. *Handbook of Graph Theory.* Discrete Mathematics and Its Applications. CRC Press, 2014. 24, 37

L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM Transactions on Graphics*, 4(2):74–123, 1985. 21, 56, 57

O. Günter and J. Lamberts. Object-oriented techniques for the management of geographic and environmental data. *The Computer Journal*, 37(1):16–25, 1994. 36

D. Hansford. The neutral case for the min-max triangulation. *Computer Aided Geometric Design*, 7(5):431–438, 1990. 52

J. Harris, J. L. Hirst, and M. Mossinghoff. *Combinatorics and Graph Theory.* Springer-Verlag New York, 2008. 24

J. H. Haunert and M. Sester. Area collapse and road centerlines based on straight skeletons. *GeoInformatica*, 12(2):169–191, 2008. 70

A. Helal, S. E. Moore, and B. Ramachandran. Drishti: an integrated navigation system for visually impaired and disabled. In *Proceedings of Fifth International Symposium on Wearable Computers*, pages 149–156, 2001. 92, 93

Ø. Hjelle and M. Dæhlen. *Triangulations and Applications.* Springer-Verlag Berlin Heidelberg, 2006. 47

K. E. Hoff, III, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 277–286, 1999. 94

H. Holone, G. Misund, and H. Holmstedt. Users are doing it for themselves: Pedestrian navigation with user generated content. In *Proceedings of Next Generation Mobile Applications, Services and Technologies, NGMAST '07.*, pages 91–99, Sept 2007. 91

O. Huisman and R. A. De By. *Principles of Geographic Information Systems.* The International Institute for Geo-Information Science and Earth Observation, 4th edition, 2009. 1, 2

W. E. Huxhold. *An Introduction to Urban Geographic Information Systems.* Oxford University Press, Inc., 1991. 35

B. Höfle, T. Geist, M. Rutzinger, and N. Pfeifer. Glacier surface segmentation using airborne laser scanning point cloud and intensity data. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(Part 3):W52, 2007. 63

ITU-T X.667. Information technology – procedures for the operation of object identifier registration authorities: Generation of universally unique identifiers and their use in object identifiers, 2012. 32, 43

B. Joe and C. Wang. Duality of constrained voronoi diagrams and delaunay triangulations. *Algorithmica*, 9(2):142–155, 1993. 56

C.-J. Jorgensen and F. Lamarche. From geometry to spatial reasoning : Automatic structuring of 3d virtual environments. In *Motion in Games*, volume 7060 of *Lecture Notes in Computer Science*, pages 353–364. Springer Berlin Heidelberg, 2011. 94

Marcelo Kallmann. Navigation queries from triangular meshes. In *Proceedings of the Third International Conference on Motion in Games*, MIG'10, pages 230–241. Springer-Verlag, 2010. 95

Marcelo Kallmann and Mubbasir Kapadia. Navigation meshes and real-time dynamic planning for virtual worlds. In *ACM SIGGRAPH 2014 Courses*, SIGGRAPH '14, pages 1–81, 2014. 94

V. Karamcheti, C. Li, I. Pechtchanski, and C. K. Yap. A core library for robust numeric and geometric computation. In *Proceedings of the Fifteenth Annual Symposium on Computational Geometry*, SCG '99, pages 351–359, 1999. 84

M. I. Karavelas. A robust and efficient implementation for the segment voronoi diagram. In *Proceedings of 1st International Symposium on Voronoi Diagrams in Science and Engineering*, pages 51–62, 2004. 56, 66, 67, 86

H. A. Karimi and P. Kasemsuppakorn. Pedestrian network map generation approaches and recommendation. *International Journal of Geographical Information Science*, 27 (5):947–962, 2013. 94

P. Kasemsuppakorn. *Methodology and Algorithms for Pedestrian Network Construction.* PhD thesis, University of Pittsburgh, School of Information Sciences, 2011. 93

P. Kasemsuppakorn and H. A. Karimi. Personalised routing for wheelchair navigation. *Journal of Location Based Services*, 3:24–54, 2009. 92, 93

L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996. 94

L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Computational Geometry*, 13(1):65–90, 1999. 21

L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom examples of robustness problems in geometric computations. *Computational Geometry: Theory and Applications*, 40(1):61–78, 2008. 83

T. H. Kolbe, G. Gröger, and L. Plümer. Citygml: Interoperable access to 3d city models. In *Geo-information for Disaster Management*, pages 883–899. Springer Berlin Heidelberg, 2005. 118

M. Laakso, T. Sarjakoski, L. Lehto, and L. T. Sarjakoski. An information model for pedestrian routing and navigation databases supporting universal accessibility. *Cartographica*, 48(2):89–99, 2013. 93

F. Lamarche. Topoplan: a topological path planner for real time human navigation under floor and ceiling constraints. *Computer Graphics Forum*, 28(2):649–658, 2009. 94

M. Lau, G. Saul, J. Mitani, and T. Igarashi. Modeling-in-context: User design of complementary objects with a single photo. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, SBIM '10, pages 17–24, 2010. 117

C. L. Lawson. Transforming triangulations. *Discrete Mathematics*, 3(4):365–372, 1972. 50

C. L. Lawson. Software for $c^1$-interpolation. In *Procedings of Mathematical Software III*, 1977. 50

R. G. Laycock and A. M. Day. Automatically generating roof models from building footprints. In *Proceedings of 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, 2003. 70

H. Ledoux and M. Meijers. Validation of planar partitions using constrained triangulations. In *Proceedings of Joint International Conference on Theory, Data Handling and Modelling in GeoSpatial Information Science*, pages 51–55, 2010. 25, 26

H. Ledoux, K. A. Ohori, and M. Meijers. Automatically repairing invalid polygons with a constrained triangulation. In *Multidisciplinary Research on Geographical Information in Europe and Beyond. Proceedings of the AGILE'2012 International Conference on Geographic Information Science*, pages 13–18, 2012. 28

D. T. Lee and B. J. Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980. 56, 57, 85, 111, 161, 162

C. Li, S. Pion, and Yap C. K. Recent progress in exact geometric computation. *The Journal of Logic and Algebraic Programming*, 64(1):85–111, 2005. 84

Y. Li, X. Wu, Y. Chrysanthou, A. Sharf, D. Cohen-Or, and N. J. Mitra. Globfit: Consistently fitting primitives by discovering global relations. *ACM Transactions on Graphics*, 30(4):52–63, 2011. 117

P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1):59 – 82, 1991. 33

Y. Liu and J. Snoeyink. A comparison of five implementations of 3d delaunay tessellation. *Combinatorial and Computational Geometry*, 52:439–458, 2005. 56

C. D. Lloyd. *Spatial Data Analysis, An Introduction for GIS Users.* Oxford University Press, 2010. 17

T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979. 94

M. P. Lynch and A. J. Saalfeld. Conflation: Automated map compilation – a video game approach. In *Proceedings of Auto-Carto VII*, page 343–352, 1985. 26

D. M. Mark. Geographic information science: Defining the field. In *Foundations of Geographic Information Science*, chapter 1, pages 1–18. Taylor & Francis, 2003. 1

H. Matijevic, Z Biljecki, S. Pavicic, and Roic M. Transaction processing on planar partition for cadastral application. In *Proceedings of FIG Working Week 2008 - Integrating Generations*, 2008. 29

B. Mayerhofer, B. Pressl, and M. Wieser. Odilia - a mobility concept for the visually impaired. In *Computers Helping People with Special Needs*, volume 5105 of *Lecture Notes in Computer Science*, pages 1109–1116. Springer Berlin Heidelberg, 2008. 92, 93

D. H. McLain. Two dimensional interpolation from random data. *The Computer Journal*, 19(2):178–181, 1976. 57

K. Mehlhorn and S. Näher. Leda: A platform for combinatorial and geometric computing. *Communications of the ACM*, 38(1):96–102, 1995. 84

K. Mehlhorn and S. Schirra. Exact computation with leda real - theory and geometric applications. In *Proceedings of Symbolic Algebraic Methods and Verification Methods*, pages 163–172. Springer-Verlag, 2001. 84

A. Mirante and N. Weingarten. The radial sweep algorithm for constructing triangulated irregular networks. *IEEE Computer Graphics Applications*, 2(3):11–21, 1982. 57

M. Molenaar. *An Introduction to the Theory of Spatial Object Modelling for GIS*. USA Taylor & Francis, 1998. 35

M. E. Mortenson. *Geometric Modelling*. Wiley Computer Publishing, 1997. 117

D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2):217–236, 1978. 21, 22

S. Musuvathy, E. Cohen, and J. Damon. Computing medial axes of generic 3d regions bounded by b-spline surfaces. *Computer Aided Design*, 43(11):1485–1495, 2011. 66

M. Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, 1988. 21, 22, 117

A. Naeve and J. O. Eklundh. Representing generalized cylinders. In *Proceedings of Europe-China Workshop on Geometrical Modelling and Invariants for Computer Vision*, pages 63–70, 1995. 119

P. Neis and D. Zielstra. Generation of a tailored routing network for disabled people based on collaboratively collected geodata. *Applied Geography*, 47:70–77, 2014. 93

Q. D. Nguyen, A. Devaux, M. Bredif, and N. Paparoditis. 3d heterogeneous interactive web mapping application. In *Virtual Reality (VR), IEEE*, pages 323–324, 2015. 117

D. Nieuwenhuisen, A. Kamphuis, and M. H. Overmars. High quality navigation in computer games. *Sci. Comput. Program.*, 67(1):91–104, 2007. 94

OGC Simple Features. OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture, 2011. 23, 25, 27, 28

B. M. Oh, M. Chen, J. Dorsey, and F. Durand. Image-based modeling and photo editing. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 433–442, 2001. 117

K. A. Ohori. Validation and automatic repair of planar partitions using a constrained triangulation. Master's thesis, Geomatics, Delft University of Technology, 2010. 25, 26

K. A. Ohori, H. Ledoux, and M. Meijers. Automatically repairing polygons and planar partitions with prepair and pprepair. In *Proceedings of the 4th Open Source GIS UK Conference*, 2012. 29

M. R. Oswald, E. Töppe, C. Nieuwenhuis, and D. Cremers. A review of geometry recovery from a single image focusing on curved object reconstruction. In *Innovations for Shape Analysis*, pages 343–378. Springer Berlin Heidelberg, 2013. 120

K. Ouchi. Real/expr: Implementation of an exact computation package. Master's thesis, New York University, Department of Computer Science, 1997. 84

M. L. Overton. *Numerical Computing with IEEE Floating Point Arithmetic*. SIAM, 2001. 83

N. Paparoditis, J. P. Papelard, B. Cannelle, A. Devaux, B. Soheilian, N. David, and E. Houzay. Stereopolis ii: A multi-purpose and multi-sensor 3d mobile mapping system for street visualisation and 3d metrology. *Revue française de photogrammétrie et de télédétection*, (200):69–79, 2012. 11, 107, 110, 146

S. H. Park, S. S. Lee, and J. Kim. A surface reconstruction algorithm using weighted alpha shapes. In *Fuzzy Systems and Knowledge Discovery, Second International Conference,FSKD*, volume 3613 of *Lecture Notes in Computer Science*, pages 1141–1150. Springer, 2005. 63

F. Penninga, E. Verbree, W. Quak, and P. van Oosterom. Construction of the planar partition postal code map based on cadastral registration. *GeoInformatica*, 9(2):181–204, 2005. 25

J. Philip. An algorithm for determining the position of a circle in 3d from its perspective 2d projection. TRITA / MAT / MA: TRITA, Royal Institute of Technology, 1997. 124

F. J. Pierce and D. Clay. *GIS Applications in Agriculture*. CRC Press, 2007. 1

L. Plümer and G. Gröger. Achieving integrity in geographic information systems - maps and nested maps. *GeoInformatica*, 1(4):345–367, 1997. 23, 29, 33

M. Pollefeys and L. V. Gool. From images to 3d models. *Communications of the ACM*, 45(7):50–55, 2002. 117

J. Ponce, D. Chelberg, and W. B. Mann. Invariant properties of straight homogeneous generalized cylinders and their contours. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(9):951–966, 1989. 120

F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, 1977. 163

T. Pylvanainen, J. Berclaz, T. Korah, V. Hedau, M. Aanjaneya, and R. Grzeszczuk. 3d city modeling from street-level data for augmented reality applications. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 238–245, 2012. 117

M. Richetin, M. Dhome, J. T. Lapreste, and G. Rives. Inverse perspective transform using zero-curvature contour points: application to the localization of some generalized cylinders from a single view. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13(2):185–192, 1991. 120

L. Ross. *Virtual 3D City Models in Urban Land Management - Technologies and Applications.* PhD thesis, Technical University of Berlin, 2010. 119

R. Safaee-Rad, I. Tchoukanov, K.C. Smith, and B. Benhabib. Three-dimensional location estimation of circular features for machine vision. *Robotics and Automation, IEEE Transactions on*, 8(5):624–640, 1992. 124, 125

C. P. Sang and C. C. Yun. Mitered offset for profile machining. *Computer-Aided Design*, 35(5):501–505, 2003. 70

H. Sato and T. O. Binford. Finding and recovering shgc objects in an edge image. *CVGIP: Image Understanding*, 57(3):346–358, 1993. 120

G. Saupin, O. Roussel, and J. Le Garrec. Robust and scalable navmesh generation with multiple levels and stairs support. In *Proceedings of 21st International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision in cooperation with EUROGRAPHICS*, number 161-170, 2013. 94

P. Sayd, M. Dhome, and J. M. Lavest. Recovering generalized cylinders by monocular vision. In Jean Ponce, Andrew Zisserman, and Martial Hebert, editors, *Object Representation in Computer Vision II*, volume 1144, pages 25–51. Springer Berlin Heidelberg, 1996. 120

S. Schirra. Precision and robustness in geometric computations. In *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 255–287. Springer Berlin Heidelberg, 1997. 84

S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 519–528, 2006. 117

A. Serna and B. Marcotegui. Detection, segmentation and classification of 3d urban objects using mathematical morphology and supervised learning. *{ISPRS} Journal of Photogrammetry and Remote Sensing*, 93:243 – 255, 2014. 11, 109

S. A. Shafer. *SHADOWS ANDSILHOUETI'ES IN COMPUTER VISION.* Kluwer, 1985. Chapter 8. 119

M. Sharir and E. Welzl. Random triangulations of planar point sets. In *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*, SCG '06, pages 273–281. ACM, 2006. 49

A. K. Sharma. *Text Book of 3D Sphere, Cone and Cylinder.* Discovery Publishing House, 2005. 125

W. Shen, J. Zhang, and F. Yuan. A new algorithm of building boundary extraction based on lidar data. In *2011 19th International Conference on Geoinformatics*, pages 1–4, 2011. 63

K. Siddiqi and S. M. Pizer. *Medical Representations*. Computational Imaging and Vision. Springer Berlin Heidelberg, 2008. 64, 65, 67

S. P. Singh, K. Jain, and V. R. Mandla. Virtual 3d city modelling: techniques and applications. *ISPRS 8th 3DGeoInfo Conference & WG II/2 Workshop, International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-2/W2:73–91, 2013. 119

G. Snook. Simplified 3d movement and pathfinding using navigation meshes. In Mark DeLoura, editor, *Game Programming Gems*, pages 288–304. Charles River Media, 2000. 94

B. Soheilian and M. Brédif. Multi-view 3d circular target reconstruction with uncertainty analysis. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Volume II-3:143–148, 2014. 138

M. Southworth. Designing the walkable city. *Journal of Urban Planning and Development*, 131(4):246–257, 2005. 92

A. Stark, M. Riebeck, and J. Kawalek. How to design an advanced pedestrian navigation system: Field trial results. In *4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, pages 690–694, 2007. 91, 93

H. Stefan. *Computing Straight Skeletons and Motorcycle Graphs: Theory and Practice*. PhD thesis, Faculty of Natural Sciences, University of Salzburg, 2011. 64, 70

S. Steiniger, M. Neun, and A. Edwardes. Lecture notes: Foundations of location based services. University of Zurich, 2006. 91

STL. StereoLithography Interface Specification, 3D Systems, Inc., October 1989. 22

I. Stroud. *Boundary Representation Modelling Techniques*. Springer-Verlag London, 2006. 117

P. Su and R. L. S. Drysdale. A comparison of sequential delaunay triangulation algorithms. *Computational Geometry*, 7(5–6):361 – 385, 1997. 56

P. Svestka and M. H. Overmars. Coordinated path planning for multiple robots. *Robotics and Autonomous Systems*, 23:125–152, 1998. 94

S. Syed, P. Dare, and S. Jones. Semi-automatic 3d building model generation from lidar and high resolution imagery. In *Proceedings of SSC Spatial Intelligence*, 2005. 117

A. Tagliasacchi. Skeletal representations and applications. *Computing Research Repository (CoRR)*, abs/1301.6809, 2013. 64, 65, 67

M. Teichmann and M. Capps. Surface reconstruction with anisotropic density-scaled alpha shapes. In *Proceedings of the Conference on Visualization '98*, VIS '98, pages 67–72. IEEE Computer Society Press, 1998. 62

D. Theobald. Topology revisited: representing spatial relations. *International journal of geographical information science*, 15(8):689–705, 2001. 19

G. Thomas and S. Donikian. Modelling virtual cities dedicated to behavioural animation. *Computer Graphics Forum*, 2000. 119

N. Tryfona, D. Pfoser, and T. Hadzilacos. Modeling behavior of geographic objects: An experience with the object modeling technique. In *Advanced Information Systems Engineering*, volume 1250 of *Lecture Notes in Computer Science*, pages 347–359. Springer Berlin Heidelberg, 1997. 36

G. Tzoumas. Exact medial axis of quadratic nurbs curves. In *Proceedings of 27th European Workshop on Computational Geometry*, 2011. 65

F. Ulupinar and R. Nevatia. Shape from contour: Straight homogeneous generalized cylinders and constant cross section generalized cylinders. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(2):120–135, 1995. 120

S. Utcke and A. Zisserman. Projective reconstruction of surfaces of revolution. In *Pattern Recognition*, volume 2781, pages 265–272. Springer Berlin Heidelberg, 2003. 120

B. Vallet and J. P. Papelard. Road orthophoto/dtm generation from mobile laser scanning. In *ISPRS Geospatial Week Laserscanning*, 2015. 11, 108

M. van Kreveld. Digital elevation models and tin algorithms. In *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 37–78. Springer Berlin Heidelberg, 1997. 47

J. Vince. *Calculus for Computer Graphics.* Springer, 2013. 112

G. Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier mémoire: Sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 133: 97–178, 1907. 53

G. Voronoï. Nouvelles applications des paramètres continus à théorie des formes quadratiques. deuxième mémoire: Recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 134:198–287, 1908. 53

V. Walter, M. Kada, and H. Chen. Shortest path analyses in raster maps for pedestrian navigation in location based systems. In *ISPRS Commission IV, WG IV / 6*, volume 31, 2006. 92

D. F. Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981. 56

K. Weiler. Edge-based data structures for solid modeling in a curved surface environment. *IEEE Computer Graphics and Applications*, 5:21–40, 1985. 21, 22

K. Weiler. The radial edge structure: A topological representation for non-manifold geometric modeling. In *Geometric Modeling for CAD Applications*, pages 3–36. North-Holland, 1988. 21

R. Wein, E. Fogel, D. Halperin, M. Hemmer, O. Salzman, and B. Zukerman. 2d arrangements. CGAL-4.6.2 User and Reference Manual, 2015. 32

C. Wiedemann. External evaluation of road networks. *International Archives of Photogrammetry and Remote Sensing*, 34:93–98, 2003. 94

T. Xue, L. Jianzhuang, and T. Xiaoou. 3-d modeling from a single view of a symmetric object. *Image Processing, IEEE Transactions on*, 21(9):4180–4189, 2012. 117

C. K. Yap and T. Dubé. The exact computation paradigm. *Computing in Euclidean Geometry*, 4:452–492, 1995. 84

M. Yirci, M. Brédif, J. Perret, and N. Paparoditis. 2d arrangement-based hierarchical spatial partitioning: An application to pedestrian network generation. In *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '13, pages 31–36, 2013. 93

J. Yu, C. K. Yap, Z. Du, S. Pion, and H. Brönnimann. The design of core 2: A library for exact numeric computation in geometry and algebra. In *Mathematical Software – ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 121–141. Springer Berlin Heidelberg, 2010. 84

S. Yuan and C. Tao. Development of conflation components. In *Proceedings of Geoinformatics and Socioinformatics*, pages 1–13, 1999. 26

M. Zerroug and R. Nevatia. Part-based 3d descriptions of complex objects from a single image. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(9):835–848, 1999. 120

# 2D Arrangements for Public Space Mapping and Transportation

This thesis addresses easy and effective development of mapping and transportation applications which especially focuses on the generation of pedestrian networks for applications like navigation, itinerary calculation, accessibility analysis and urban planning. In order to achieve this goal, we proposed a two layered data model which encodes the public space into a hierarchy of semantic geospatial objects. At the lower level, the 2D geometry of the geospatial objects are captured using a planar partition which is represented as a topological 2D arrangement. This representation of a planar partition allows efficient and effective geometry processing and easy maintenance and validation throughout the editions when the geometry or topology of an object is modified. At the upper layer, the semantic and thematic aspects of geospatial objects are modelled and managed. The hierarchy between these objects is maintained using a directed acyclic graph (DAG) in which the leaf nodes correspond to the geometric primitives of the 2D arrangement and the higher level nodes represent the aggregated semantic geospatial objects at different levels. We integrated the proposed data model into our GIS framework called StreetMaker together with a set of generic algorithms and basic GIS capabilities. This framework is then rich enough to generate pedestrian network graphs automatically. In fact, within an accessibility analysis project, the full proposed pipeline was successfully used on two sites to produce pedestrian network graphs from various types of input data: existing GIS vector maps, semi-automatically created vector data and vector objects extracted from Mobile Mapping lidar point clouds.

While modelling 2D ground surfaces may be sufficient for 2D GIS applications, 3D GIS applications require 3D models of the environment. 3D modelling is a very broad topic but as a first step to such 3D models, we focused on the semi-automatic modelling of geospatial objects (such as poles, lampposts, tree trunks, etc.) which can be modelled or approximated by generalized cylinders from single images. The developed methods and techniques are presented and discussed.

**Keywords :** *2D Arrangements, Alpha shapes, Centreline generation, Computational geometry, Delaunay triangulations, Directed Acyclic Graphs (DAG), Geographic Information Systems (GIS), Medial axis, Hierarchical GIS modelling, Pedestrian network graphs, Planar partition, Robust exact geometric computation, Straight skeleton, StreetMaker, 3D generalized cylinder modelling*