INVERSE PROCEDURAL STREET MODELLING: FROM INTERACTIVE TO AUTOMATIC RECONSTRUCTION

RÉMI CURA

École Doctorale MSTIC

Ingenieur-Docteur
École Doctorale MSTIC
Université Paris Est Marne la Vallée
Avril 2016 – version 1.00

Figure 1: Japan streets viewed from Osaka Big Wheel, ©Romane Gauriau 2015.

Rémi Cura: *Inverse procedural Street Modelling: from interactive to automatic reconstruction*,
École Doctorale MSTIC, © Avril 2016
SUPERVISORS:
Nicolas Paparoditis
Julien Perret
LOCATION:
Paris
TIME FRAME:
Avril 2016

Figure 2: Republique place, terrestrial Lidar and street view separated by a year. Urban space use has already been changed.

RC:0.0.0.0. dedication : fill dedication before final submission

RC:0.0.0.0. general : add a flip book illustration

RC:0.0.0.0. general : add a graphical table of figures

This thesis manuscript is a lengthy read touching up on widely different subjects, such as point cloud management and level of details, procedural modelling, user interaction and non linear least square optimisation.

For each chapter, a special "Thesis thread" section introduces it and explain its global role in the thesis.

Those "Thesis thread" sections have a different background colour for immediate identification.

This thesis is designed to offer several faster alternatives to full read.

Very fast Read

Only read Abstract (5 on page 269) and the Graphical Abstract (Figure 4 on page 5)

Independent read

this manuscript was designed to be split into five chapters that can be read independently.

Read one chapter independently (chap 1 on page 8, chap 2 on page 51, chap 3 on page 88, chap 4 on page 122, chap 5 on page 148).

Nevertheless we recommend reading Abstract (5 on page 269) and Introduction first (0 on page 2).

Fast read

read the Abstract (5 on page 269) and introduction (0 on page 2), the two pages summary for each chapter *two pages summary* (chap 1 on page 8, chap 2 on page 51, chap 3 on page 88, chap 4 on page 122, chap 5 on page 148), and the conclusion (6 on page 188).

Fast graphical read

Figure 4 on page 5 is the general graphical abstract of the thesis. Each chapter has a graphical abstract on its second page.

This thesis contains a graphical table of figures (Page 247), where miniatures of all figures are listed by chapter, along with a link and page number of the original figure.

The graphical table of figures is page 247. For each chapter, the first figure is this chapter graphical abstract.

PUBLICATIONS

Presented

Cura, 2014a

Published

Cura, Perret, and Paparoditis, 2015b
Cura, Perret, and Paparoditis, 2015a

Accepted

Working Paper

Cura, Perret, and Paparoditis, 2016

ACKNOWLEDGEMENTS

RC:0.0.0.0. Acknowledgements: to be filled after final review. Put ANRT grant reference

VOCABULARY DISAMBIGUATION

We briefly precise the meaning we give to the vocabulary used throughout this manuscript.

CITY We use "city" as its most common acceptation, that is a densely populated and socially complex place fulfilling certain functions to its surroundings. For this thesis cities also contains buildings and streets. We do not consider other public space (garden, place, etc.).

STREET We consider that streets are extremely important components of a city. Streets have a physical aspect (geometry, colour, etc.) but also a semantic aspect (name, organisation, usage, relation between constituents, etc.). In particular, streets contains street objects (or street features), such as trees, markings, urban furnitures, etc. We always consider that a street is organised around a road. Therefore describing a street requires describing the underneath road.

ROAD Similarly to streets, roads have a physical and semantic aspect. The physical part is the roadway, that is the road surface vehicles are using. In city, roadway is limited by sidewalk. Roads semantic aspect is related to how those roads are used, and so are related to traffic information (lanes, information about allowed trajectories, etc.) for vehicles, and pedestrian crossing for pedestrian.

KERB, CURBSTONE, SIDE-WALK A side-walk as a close relation to roads, yet has a much more diverse utilisation (for instance terraces may be set on it.) The sidewalk is often described as the surface complementary to roads and buildings, which we consider partially false (counter example : place, garden). Instead, the sidewalk is a medium that is dedicated to interface public and private space in a city, while also allowing pedestrian traffic. The sidewalk is separated from the roadway by curbstones or kerb, which have a leading role in this thesis, as they structures streets.

MODEL VERSUS MODELLING "Model" is a slippery word, with different meanings in social science, physic, computer science, etc. We try to differentiate between model and modelling, where the model is an organisation, and modelling is an actual instance of the model. For instance in Chapter 5, we use a simple road model that is constituted of a road axis network (polylines) associated with a road width. The road modelling is then the actual linestring with actual road width value that forms an actual (implicit) road surface. Similarly, in Chapter 3, we model street objects as semantic points whose position and orientation is defined regarding a road axis. The modelling of such a street object would be an actual semantic label (such as "public light"), associated with actual values of curvilinear abscissa and distance to road axis, along with orientation relative to road axis orientation.

USER INTERACTION/ INTERACTIVE Chapter 4 is dedicated to user interaction, so that user can interactively edit the street modelling. By "user interaction", we mean classical user input through a graphical user interface (any GIS software with read/write capabilities to PostGIS in our case).

We use two meanings for "interactive". Most often, we mean that the interactive loop feedback (time between a user input and the end of the triggered reaction) is less than 0.3 ms, which is an estimate of the threshold for instantaneous perception of a complex visual situation (Below this threshold, the user has the impression that the reaction was instantaneous). However we relax this definition by considering that occasional reaction times of a few seconds are still acceptable for a good quality user interaction. For instance, in StreetGen (Chapter 3), common edits like editing a road direction is under 0.3 ms, which makes them appear instantaneous. On the opposite, editing several trajectories at the same time in a very large intersection is less than a few seconds, which is acceptable because this operation is not frequent.

DATABASE Almost all the work of this thesis is performed inside a PostgreSQL Relational database management system. We often shorten this server designation to "the database". We stress that a database is not a set of data, but also relation between data, and a set of integrated tools, as well as a dedicate data-retrieval language (SQL).

GENERALISATION We use the concept of generalisation in Chapter 2 and Appendix 7. We consider the generalisation as the process to make data more general, that is abstract , simplify and synthesize of geospatial data for visualisation or other applications. We refer to (Burghardt, Duchêne, and Mackaness, 2014, Chap. 1) for a more comprehensive definition.

CLASSIFICATION AND CLASS We analyse point cloud classification in Appendix 7. We use the common acceptation, that is that classification is the process to provide for each point a label referring to predefined classes. For instance, after classification, a point might have a label 'tree', 'ground' or '2-wheelers'.

CONTENTS

Part I

INTRODUCING THE THESIS

0

A GENERAL INTRODUCTION TO THIS THESIS

GENERAL INTRODUCTION

Context

Fast growing cities

World population is growing fast. A recent survey (United Nations, 2012) shows that 52% of Mankind already lives in urban area. These urban areas are expected to absorb more than the demographic augmentation, with new cities reaching the million of inhabitants every year in Africa and Asia. The cities not only grow by number of inhabitants but also by the area they occupy. The urban land use is expected to increase in the order of 100 000's km² in the next decade (Seto et al., 2011).

With population concentration, tensions are building up

While the number of cities is growing, cities are also getting bigger : 40% of city inhabitants are living in cities over 1 million inhabitants. The growth of cities is partially absorbed by the constitution of megacities. 10% of world population live in megacities (23 cities that are bigger than 10 millions), and this should increase to 13.5 % until 2025 (United Nations, 2012)

Cities also concentrate inequalities, which are rising in the country where urbanisation is expected to be the most significant (OECD, 2010, p.37).

High densities in cities imply careful management of the environment of a city. The necessary fluxes (in and out) are massive. Although some of thoses fluxes are natural, they are also heavily impacted by cities (water, air, heat, etc.).

With city being such concentration spots, crisis management is much more difficult, as any problem might impact a great number of people very fast (flood, power cut, epidemic, toxic dispersion, etc.). Morevover the sheer complexity of cities also complicate crisis management planning.

A dire need for urbanism

For about one century the field of urbanism has been dedicated to tackle those problems. Traditional tools from architecture and social science have been traditionally used, yet the mere size and complexity of modern city requires more and more complementary approaches.

Stakes

Computer science for urbanism

Advances of computer science and engineering have provided urban planners with simulation tools to model behaviours and even test planning scenarios. For instance a redesign scenario of an urban area can be tested with traffic simulation, economic simulation, virtual reality, etc. More than simulation, communication is also an important aspect of city planning, as planning is spread across several entities, public or private, as well as the inhabitants. Moreover, a good representation of situations and scenarios is essential for the decision process as well as for the elaboration of the planning.

A new tools: city model

Planning, communication, simulation, all those require a precise city model, in the form of a 3D model including several levels of information. This 3D model is also needed to several other applications related to cities (see (Niggeler, 2009), and Figure 3), such as transport, energy, security, entertainment, communication, geomarketing, etc.

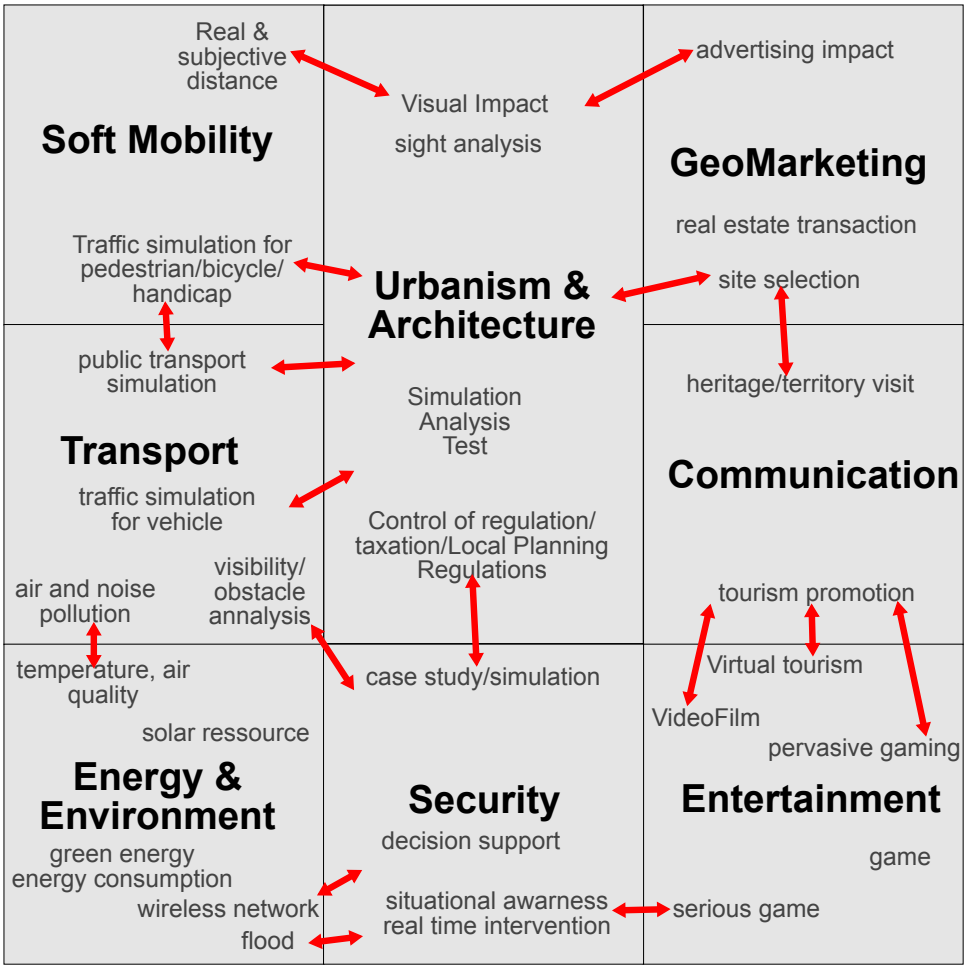


Figure 3: Some example of 3D City models usage (from (Niggeler, 2009))

Challenge

Modelling a city is complex, essentially because the city can be very large, yet contain small and important details. For instance the kerb that separate road surface and sidewalk is less than 10 cm high, yet it structures the whole street. Moreover, a city is constituted of intertwined and interdependant layers of informations, possibly (sometimes partially) following potentially vague design guidelines. To be useful, a city model has to be structured, so its information can be leveraged by other numeric methods.

With building life spawning dozens of years, and the extremely high cost of public work, one could be suprised that cities are nonetheless perpetually changing. Being massive and changing so frequently, city models can hardly be fabricated manually (or with prohibitive costs), but need to be generated automatically or semi-automatically.

Goal

The goal of this thesis is to reconstruct the streets of a city. The main guiding principle is to generate a procedural generic model and then to adapt it to reality using observations. Because streets are a defining element of cities, we first analyse how other city features (road network, streets, vegetation, urban objects) are reconstructed through the literature (Chapter 1).

In our framework, a "best guess" road model is first generated (Chapter 3) from very little information (road axis network and associated attributes), that is available in most of national databases. This road model is then fitted to observations by combining in-base interactive user edition (using common GIS software as graphical interface)(Chapter 4) with semi-automated optimisation (Chapter 5). The optimisation approach adapts the road model so it fits observations of urban features extracted from diverse sensing data. Both street generation (StreetGen) and interactions happen in a database server, as well as the management of large amount of street Lidar data (sensing data) as the observations using a Point Cloud Server (Chapter 2 and Appendix 7).

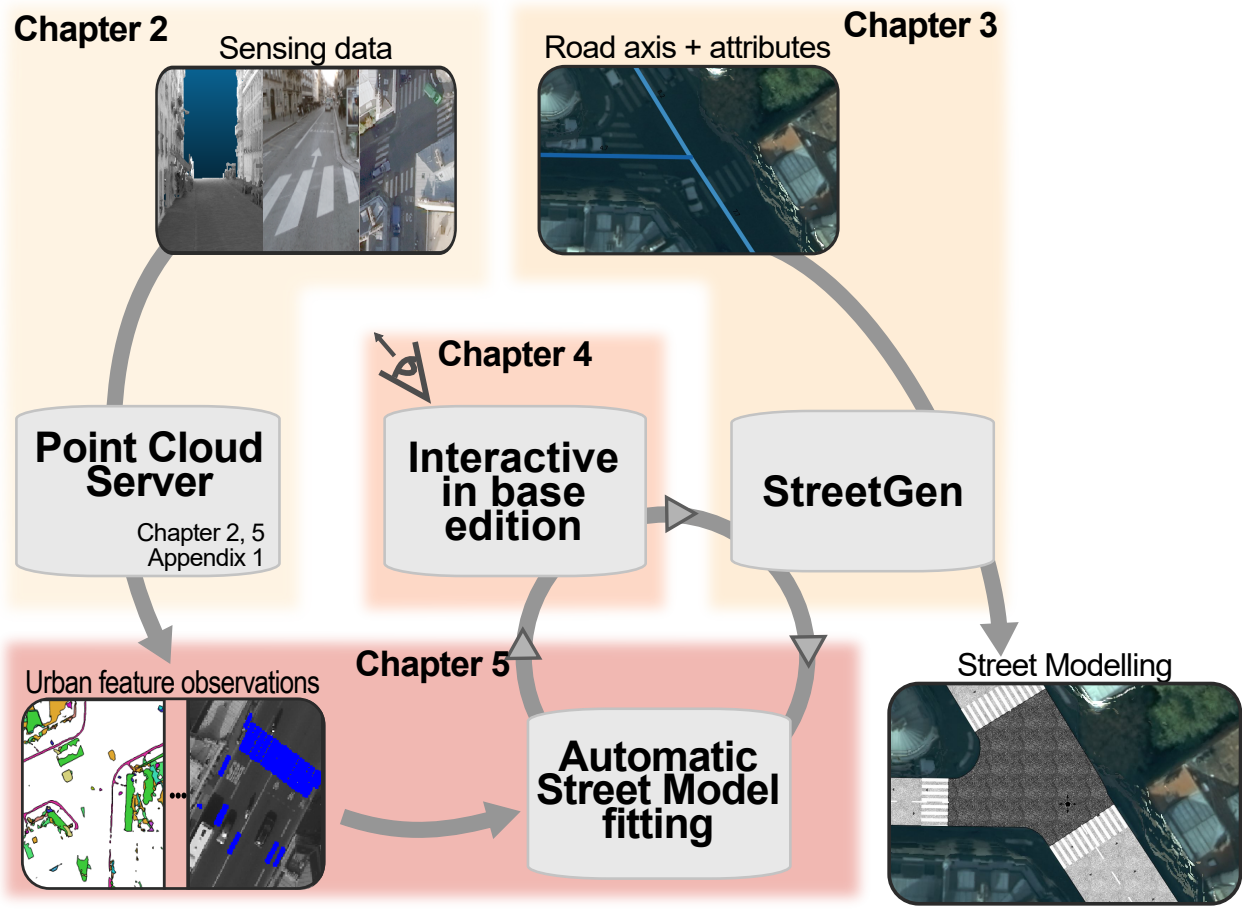


Figure 4: Thesis graphical abstract. From an approximate road axis network a street modelling is generated with StreetGen (Chapter 3). Such modelling is interactively edited with any GIS using Interactive In Base edition (Chapter 4). In complement, the street modelling can be automatically fitted (Chapter 5) to massive sensing data, managed with a Point Cloud Server (Chapter 2, Appendix 7) .

Part II

THESIS

1

URBAN RECONSTRUCTION: A STATE OF THE ART

THESIS THREAD

The goal of this thesis is to reconstruct (model) streets based on both user inputs and observations from sensing data. We face three main issues for a thorough state of the art. The first issue "seldom researched" is that although many urban features reconstruction have been well researched, we could not find research dedicated to street reconstruction. The second issue "transportation" is that streets also have a strong role to play in transportation, and thus contain roads, which are parts of a global network. The last issue "contains other features" is that by nature streets are composed of essential other urban features (street furniture, markings, etc.), as such reconstructing streets also involves reconstructing these urban features.

Due to "seldom researched", we can not do a simple state of the art of street reconstruction, and so we have to look at reconstruction methods applied to other subject for inspiration. We put a particular emphasize on road and road network reconstruction due to "transportation". Last we analyse methods for urban feature reconstruction, be it natural (vegetation) or man made. Except buildings, all aspects of urban reconstruction are then covered, and thus this state of the art is not only about street reconstruction but about urban reconstruction.

RC:1.1.0.0. chap1:abstract : insert graphical abstract

ABSTRACT

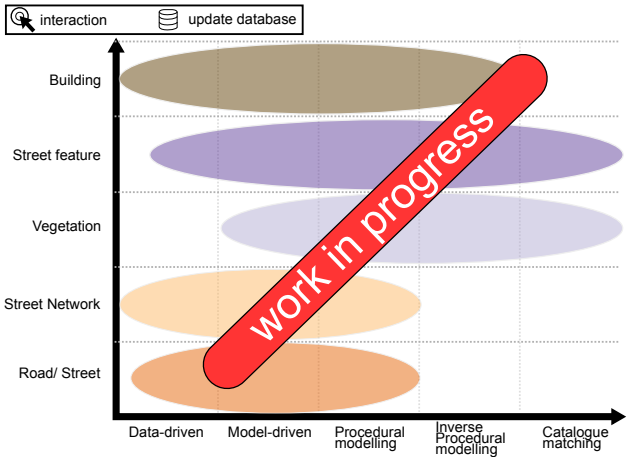


Figure 5: A subjective transverse classification of reconstruction methods by role of model.

World population is raising, especially the part of people living in cities. With increased population and complex roles regarding their inhabitants and their surroundings, cities concentrate difficulties for design, planning and analysis. These tasks require a way to reconstruct/model a city.

Traditionally, much attention has been given to buildings reconstruction, yet an essential part of city were neglected: streets. Streets reconstruction has been seldom researched. Streets are also complex compositions of urban features, and have a unique role for transportation (as they comprise roads). We aim at completing the recent state of the art for building reconstruction (Musialski et al., 2012) by considering all other aspect of urban reconstruction. We introduce the need for city models (Sec. 1.3 on the following page). Because reconstruction always necessitates data, we first analyse which data are available (Sec. 1.4 on page 17). We then expose a state of the art of street reconstruction (Sec. 1.7 on page 28), street network reconstruction (Sec. 1.8 on page 31), urban features reconstruction/modelling (vegetation (Sec. 1.9 on page 35), and urban objects reconstruction/modelling (Sec. 1.10 on page 40).

If you read the thesis introduction, you can skip this chapter introduction until Section 1.3.6 on page 14.

Context

Total population living in cities is growing

World population is increasing fast. A recent survey (United Nations, 2012) shows that 52% of Mankind already lives in urban area .

These urban areas are expected to absorb more than the demographic augmentation, with new cities reaching the million of inhabitants every year in Africa and Asia.

The cities not only grow by number of inhabitants but also by the area they occupy. The urban land use is expected to increase in the order of 100 000's km² in the next decade (Seto et al., 2011).

Tensions are building up

DEMOGRAPHIC PRESSURE. : CONCENTRATION OF POPULATION While the number of cities is growing, cities are also getting bigger : 40% of city inhabitants are living in cities over 1 million inhabitants. The growth of cities is partially absorbed by the constitution of megacities. 10% of world population lives in megacities (23 cities that are bigger than 10 millions), and this should increase to 13.5% until 2025.

SOCIAL PRESSURE Cities also concentrate inequalities, which are rising in the country where the urbanisation is expected to be the most significant (OECD, 2010, p.37).

ENVIRONMENTAL PRESSURE High densities in cities imply careful management of environment of a city. The necessary fluxes (in and out) are massive. Although some of these fluxes are natural, they are also heavily impacted by cities (water, air, heat, etc.).

CRISIS MANAGEMENT Concentration also makes crisis management much more difficult. Natural hazard (flood, earthquake, power cut) have more potent effects when hitting a city as they concern more people, and as the very density and complexity of city infrastructures might leave them more vulnerable. Cities growing very fast may outgrow their infrastructures.

Moreover, cities importance also makes them more susceptible to human-related hazard (epidemic, toxic dispersion).

Need for urbanism and city planning

For about one century the field of urbanism has been dedicated to tackle those problems. The new challenges and the change of scale of the problem necessitate new tools.

Stakes

Need for city modelling

Urbanists traditionally use methodologies from social sciences. The advance of computer science and engineering has given them simulations tools to model behaviours and even test planning scenarios. Planing is spread across several entities, public or private, as well as the inhabitants. This makes communication an important aspect of city planing. Moreover, the representation of situations and scenario is essential for the decision process as well as for the elaboration of the planning.

A new tools: the city model

2D maps have been the tool of choice, and can now advantageously be completed by structured 3D city model created from various information.

Answer part of the needs

This new model and the 3D nature brings in turn several new applications (see (Niggeler, 2009) (fr. and ge.). The Figure **??** (inspired from (Niggeler, 2009)) gives an overview of some applications for a city model.

Applications

Urbanism-related applications

(See Fig. 6) City modelling is widely used for urban planning and understanding. Having detailed city models is an asset for visualisation and simulation, permitting to test planning scenarios (new and transformation), analysing various impacts and properties (noise, pollution, light propagation, flood, power cut, epidemic, toxic dispersion, water management, temperature), or design transportation system.

Being a place of spatial and social concentration, a city is very sensible to environment issues. Monitoring and simulating air quality (Moussafir et al., 2013), temperature, wind speed, solar exposition, water cycle and so is important both for social reasons (perceived cleanness, perceived lightness), for energetical reasons (urban heating or cooling), as well as for health (being of high density, cities are more prone to epidemics).

Cities models are also used for tourism and communication as a part of the larger Virtual Reality (VR) trend. Similarly, digital mapping is used as a simpler VR application, permitting to help in a GPS-based navigation system, or simply browse pictures of the roadside.

3D model for entertainment

(See Fig. 6) The important place the cities have in our lives logically pervades into the collective images used by the entertainment industry.

Thereby many films pictures real or imagined cities, in particular to support special effects. The game industry needs are even bigger, partly because the recent trend toward Massively Multiplayer Online Role-Playing Game (MMORPG). Such games often

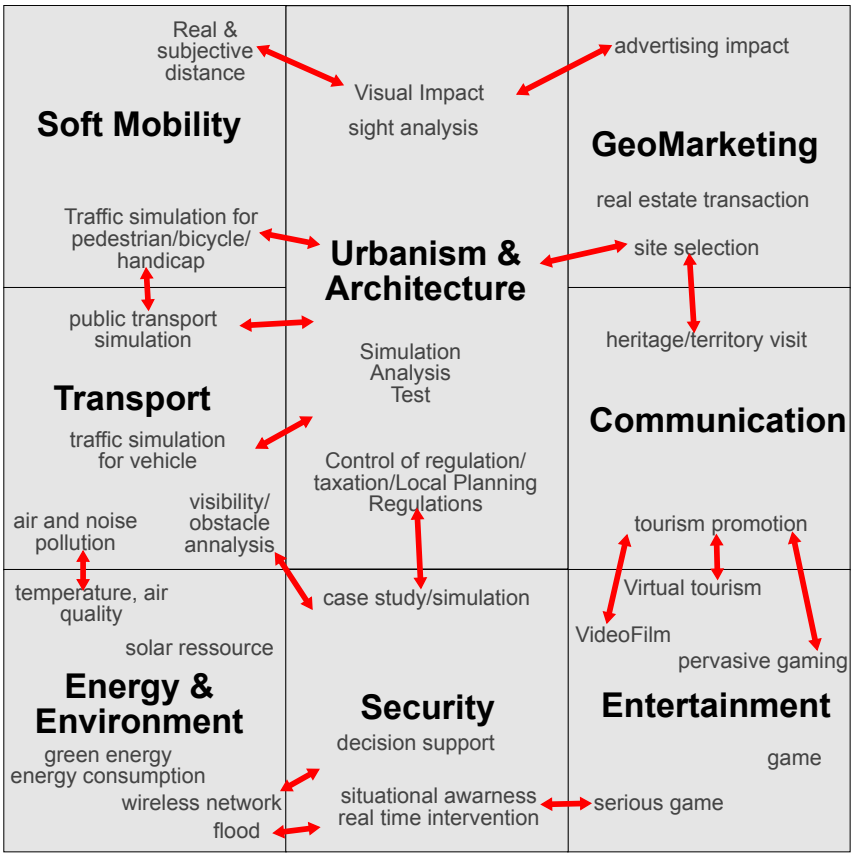


Figure 6: Example of potential usages of a city model.

induce massive open cities with believable animated agents. Interestingly, the city simulation games are similar in nature to serious training tools used to prepare emergency response, crisis management and police deployment.

What is a city

Defining properly a city is difficult, as it would involves historical and social criterias. In this work, we consider that a city is a densely populated and socially complex place fulfilling certain functions to its surroundings.

Challenges

Reconstructing 3D city model is vital for many applications that are necessary to manage and plane cities, easing the life of Billions of citizens.

Reconstructing a multi-level complex of objects

Some of the reasons that make city modelling particularly challenging can be derived from the definition we used for city: A city is a set of connected components interacting with each other. In this way the road network usually influences building placement. The social complexity accounts for various uses of space and therefore various types of

buildings (constructed for various usage and through the time). The dense population use multi modal transportation networks which share part of space (e.g. bicycle and cars).

Such a layered nature incite us to decompose the city reconstruction problem into connected problems: the reconstruction of buildings (which we defer to (Musialski et al., 2012)) , the reconstruction of streets (Section 1.7) , the reconstruction of street network (Section 1.8) , the reconstruction of urban vegetation (Section 1.9) and the reconstruction of urban objects (Section 1.10).

Multi scale

The spatial extend of a city model is large (typically in the order of the 100 to 10000km²), yet many important part of the design are small (e.g. a curb is around 0.1m high but strictly defines radically different space usages: pedestrian vs vehicle).

Automation

Manual modelling as been the tool of choice for a long time but can only be applied to small parts. Therefore city reconstruction must use automatic or interactive tools. Yet the more automatic a process is, the more it relies on data quality, which is particularly problematic in an environment where aerial data is of reduced use and land data is heavily occluded.

Cluttering

Urban environment is so dense and cluttered that usually the data is only partial (e.g. a tree hides a part of a building facade). This is illustrated in figure 7.

Figure 7: On this street Lidar point cloud, trees are clearly masking building facades, creating occlusion.

Many object categories

Cities contains many objects ("object" being used in a wide acceptance) that forms complex patterns of relations. For instance streets markings follow complex rules that enforce the highway code.

City reconstruction/modelling

City reconstruction

Several research communities have been interested in city reconstruction. It has proved to be a challenging and highly interdisciplinary set of problems, with many major practical applications.

One could think the upper bound of city reconstruction problem is to have partial models of how an existing city looks as well as how it works, sometime trough time.

However understanding the functioning of a city is out of our scope and this state of the art focus on reconstructing its physical components.

The expression "city reconstruction" and "city modelling" are used alike to designate our problem by different research communities. Both convey the same idea of a partial view of reality, along with some knowledge about its structure (reconstruction) and/or behaviour (modelling). More practically, the model is designed to be browsed through digital imaging (which according to (Ramilo, 2005), is more efficient that a real life scaled model). To be precise usually the goal is not exactly city reconstruction but more the reconstruction of a specific urban space with some of its structuring properties.

A link between modelling and reconstruction

The term "modelling" seems to be more used in the Computer Graphic community, while the term reconstruction seems to be more employed in the Photogrammetry and Remote Sensing community.

We use both because there is a point of convergence.

Trying to reconstruct a city and its components always involves an implicit model (or hypothesis) determined by the choices of algorithms and constraints (e.g. most of the time building are implicitly considered like locally planar blocks, mostly vertical). We observe a trend in reconstruction to use more abstract knowledge, like semantic consideration (e.g trying to reconstruct buildings parts with traditional stereo reconstruction as well as primitive fitting (Lafarge et al., 2010)).

In the same time, while modelling is not exclusively dedicated to depict real world objects, it is still possible to use modelling methods to get a model as close to the reality as possible. It has traditionally been done with human feedback (a 3D artist uses pictures of an object and dedicated software to draw it in 3D), but a recent trend tries to do it automatically: the inverse procedural modelling paradigm (See Sec. 1.5.2 on page 26) .

The consequence is very important for this work. Indeed most modelling methods have the potential to be used in reconstruction process via the inverse procedural modelling paradigm, and thus we include these modelling methods in this state of the art.

Scope

In this article we focus on the reconstruction of the morphological characteristics of a city.

GEOMETRICAL MODELS we focus on the methods to obtain models of cities. We limit the possible usages and data type to the most common. For instance we do not describe

audio feature, even if it is an essential piece for realism (that can also limit the need for visual details (Mastoropoulou et al., 2005)). We focus on geometrical models.

NOT ONLY BUILDINGS Most of the works in city reconstruction have been focused on buildings reconstruction (Klavdianos, Zhang, and Izquierdo, 2013; Musialski et al., 2012). Yet a city is far from being only an aggregate of buildings. Paris, one of the densest city in the world, is a good example. About 70% of the surface is *not* occupied by buildings. The streets occupy 40% and the places 5%.

We can explain this by the fact that a city is by definition a place of complex social interactions, therefore a need for a common medium is essential and must exist: the road network and the streets.

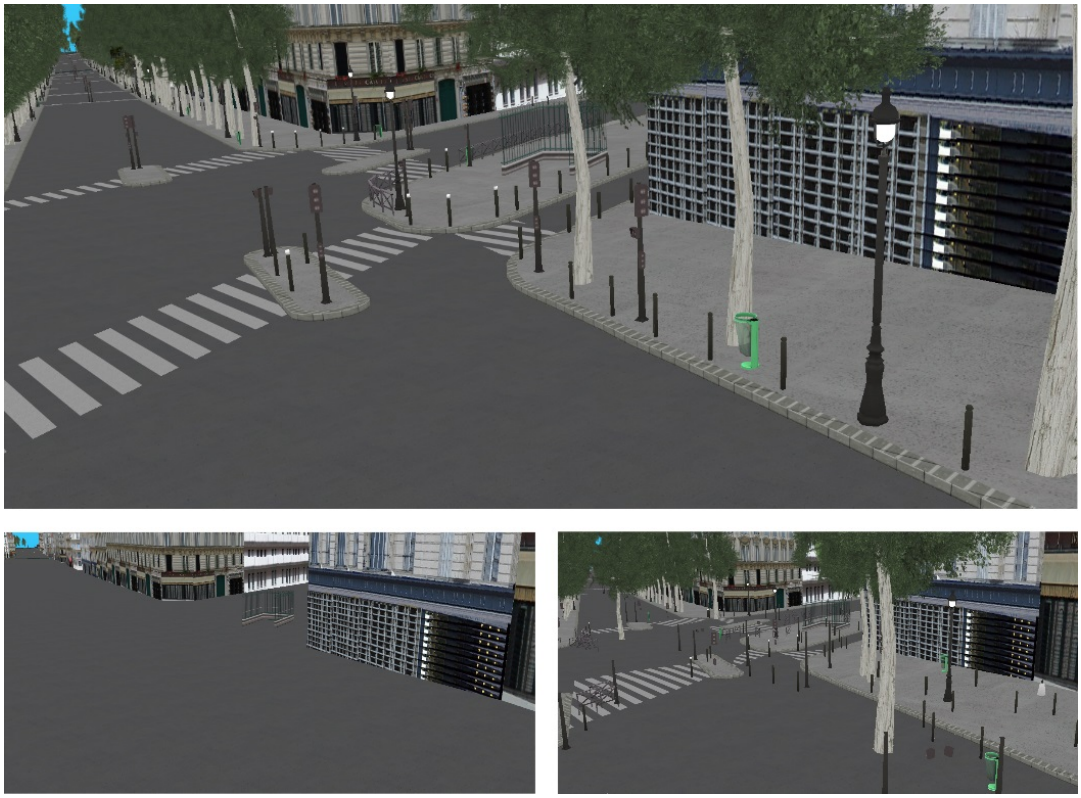


Figure 8: Synthetic 3D city model. When urban features are coherent (top), the model is a great deal more realistic than without any objects (bottom left), or even with the same amount of objects but un-organised (bottom right).

Moreover, a crude perceptive example (Figure 8) shows the importance of de road network, street, vegetation, urban objects.

A real street view of Toulouse city (Fig 9 on the following page) with objects highlighted shows the diversity and importance of street objects. Almost all the applications of city modelling (See Figure 3 on page 3) benefit as well from such additions. Another clue of the importance of non-building elements for city modelling can be given by analysis of City GML (Kolbe, Gröger, and Plümer, 2005), the leading current standard to represent city. Building is only one City GML module among a dozen other (transportation, vegetation, urban furniture ...).

Figure 9: importance objets

NO TRANSPORT SIMULATION Crowd and traffic simulation is out of the scope of this article. However, such simulation necessitate reconstruction of specific data which we will briefly cover. Reader can refer to the recent state of the art of (Duives, Daamen, and Hoogendoorn, 2013) for more details about traffic simulation.

Plan

Reconstruction always necessitates data representing the city, we first analyse which data are available (1.4).

City is composed of many components, and many methods from different research community try to reconstruct them. We first propose a trans-components classification of reconstruction approaches (1.5).

We then dress independent state of the art for each category of city components being reconstructed, such as street reconstruction (Section 1.7 on page 28), street network reconstruction (1.8 on page 31), urban features reconstruction/modelling (vegetation (1.9 on page 35), and urban objects reconstruction/modelling 1.10 on page 40).

This simple ordering is necessary due to the wide differences between methods. For each of this categories, we propose several ways to classify the state of the art methods, to allow a multi-level understanding of the field.

We conclude this work by giving perspectives about the evolution of city reconstruction.

We analyse what data are available for city components reconstruction. These data can be ordered from less structured (Lidar data, vector data) to more structured (image data, raster data), and from less abstract (Lidar data, image data) to more abstract (raster data, vector data). (See Figure 10)

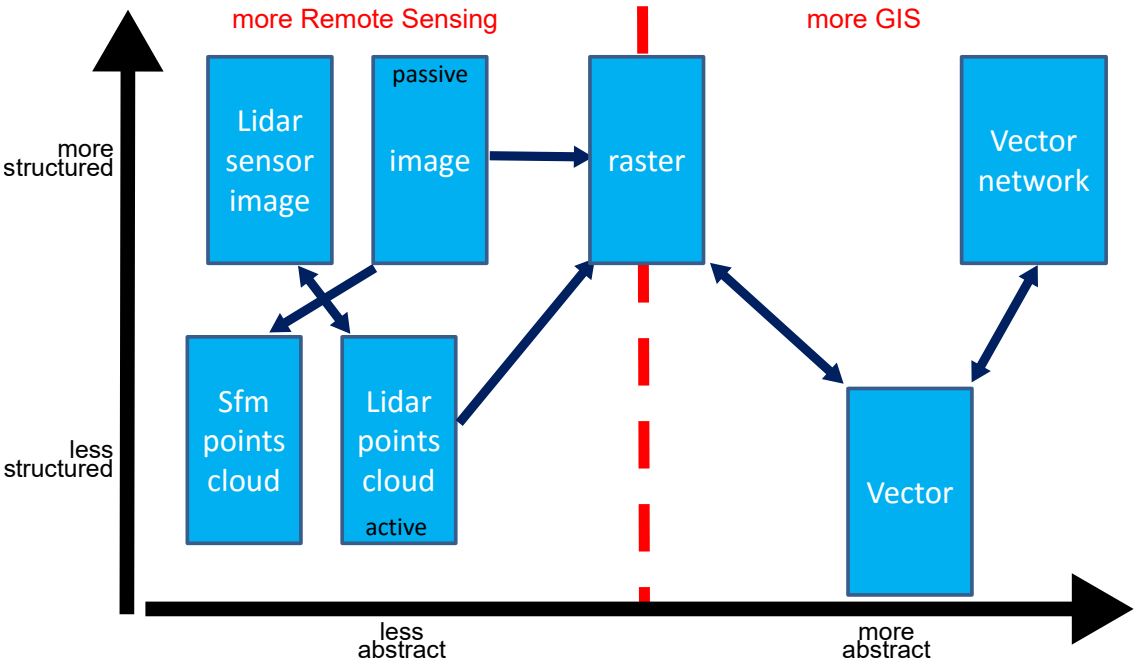


Figure 10: Available data types for city components reconstruction, ordered from less to more abstract, and from less to more structured.

Please note that this classification is based on common usage rather than on strict mathematical differences. From a mathematical point of view, Lidar, image and raster data are of same nature (2D lattices, i.e. regularly sampled values), and the definition of vectors data is vague (parametrized shapes with semantic, the types of shape and parameters varying).

Moreover the boundaries may be fuzzy. For example it is possible to create an image from a Lidar pointcloud (sensor view, See Fig. 12 on page 19), and a pointcloud from multi-images (Structure From Motion, SFM). Similarly, a conservative two-way conversion between raster and vector data is possible under certain assumptions (2.5D).

In the next sections we introduce each of this available data. Each section is illustrated by real data for a street of Toulouse city (France), from mobile mapping (Paparoditis et al., 2012), or from aerial images (French mapping agency, IGN).

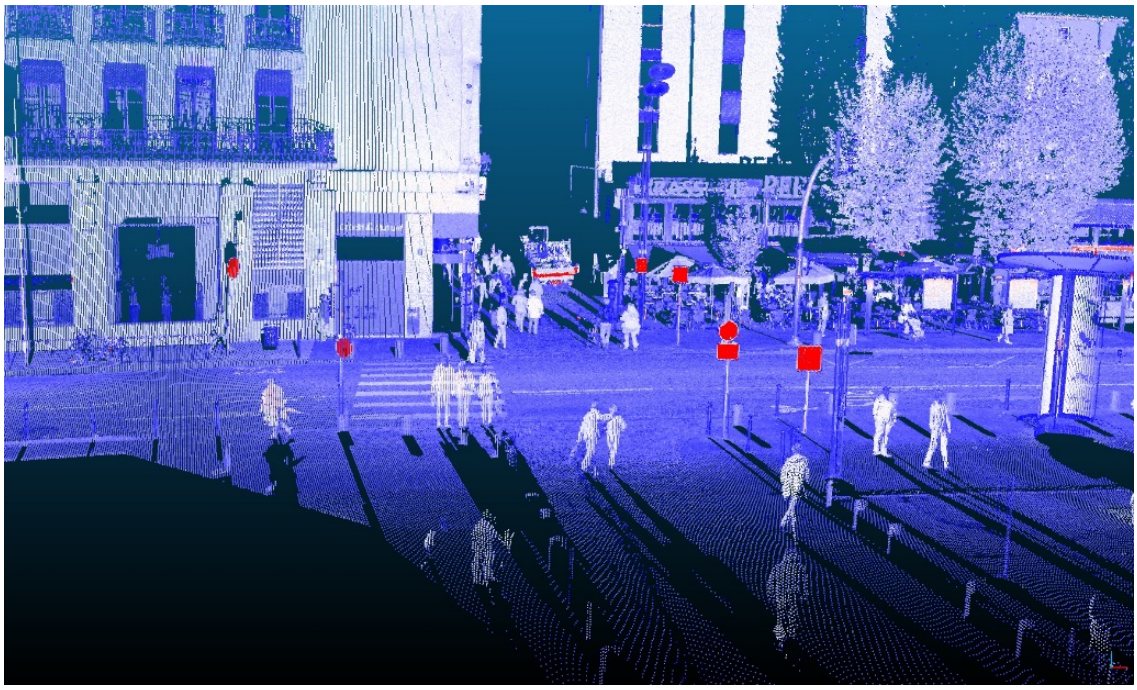


Figure 11: Street Lidar Point cloud (intensity tone from blue to white to red).

Lidar data

Intro

Light Detection And Ranging (LIDAR, noted "Lidar" for readability) data are becoming more and more available to the point of being common. Their interest is partly due to the complementary nature they offer to images, making them extremely popular in urban reconstruction. A Lidar device uses active sensing, and can be fix or embedded on mobile objects (plan, drone, vehicles, train, etc.). Figure 11 illustrates a point cloud from a terrestrial Lidar.

Principle

The Lidar principle is simple and very similar to a Laser measuring tape. The device emits a short light impulsion (i.e. active sensing) from a know position in a known direction at a precisely known time. This light signal flies for some time, hits an obstacle and is partially reflected backward to the device. The device receives this reflected signal. Then it analyses the time of flight, and given the speed of light in air, it can compute the distance from the device to the obstacle. This gives the precise 3D position of the obstacle, hence a 3D point.

The magnitude (i.e amplitude of signal) of the return impulse is also extracted, quantifying the ability of the obstacle to efficiently reflect light (at the Lidar frequency, for a given input angle). Intuitively, a street furniture of polished metal will reflect much more light than a rugged stone wall.

Data volume

Such sensing is made at high frequency (0.1 to 1 Mpts/s), making the data volume huge and barely tractable in practice. Yet, even at several millions of point per second, we are short of a typical HD video-film acquisition data rate (1200*1800 pixels, 25 times per second). However, the data volume is much more difficult to manage with Lidar data than image data (See Chapter 2 for more details about point cloud management).

One has to remember than photography have been invented two centuries ago, and that digital imaging has been researched for several decades. In opposition dense 3D point clouds and Lidar processing are much newer. The industry still lacks standard format, powerful viewers and editors, and mature compression (for example : (Mongus, rupnik, and Zalik, 2011). The link to the compressive sensing theory (Baraniuk et al., 2011) does not seem to have attracted much interest either).

All in all, the main issue with Lidar data is not its volume, but the lack of management framework as a whole (See Chapter 2).

Details and facts

Lidar can be airborne (several points per square meter, precision of 0.1 to 1m) or ground based (for stationary station: precision less than 1mm, for vehicle: precision around 0.1 to 1cm).

More sophisticated methods allow to acquire and store the full waveform of the return signal, which can be used to extract multiple points per waveform (e.g. one point for the forest canopy, one point for the forest mid level and one point for the ground) (Mallet, 2010). Other recent technologies propose multi-spectrums Lidar (Hakala et al., 2012; Wallace, Nichol, and Woodhouse, 2012), but these remain in laboratories for the moment.

It is important to note that LIDAR data is an accurate and sparse sampling of 3D objects by nature (e.g. the size of error is small compared to the distance between points). As such, their sparse and 3D nature is complementary to the high density and 2D nature of images.

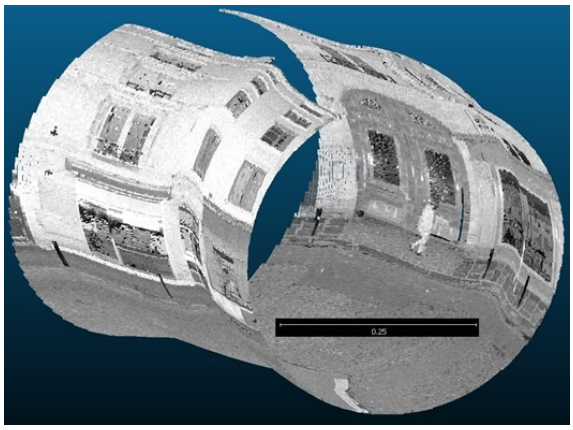


Figure 12: Lidar points can be seen as images (sensor image) as the acquisition process is regular.

It is possible to create an image from the Lidar device point of view (sensor image, see Figure 12), because the device physically acquires points following a very regular

pattern (lines, grids). Such an image is equivalent (dual) to the traditional 3D point cloud, and can be used as a 3D depth map. However it is not commonly used because working in image space disables the possibility to use classic Euclidian distances. There is no straight relation between 3D distance and pixel distance (2 points separated by 6 meters in 3D world may be separated by any number of pixels).

Images

Figure 13: Street View (360° panoramic) .

Introduction

Images are very common data, partly because they are widely available and have been used in computer science for a long time. Images can be aerial images (See Figure 14) or street view images (See Figure 13). Efficient image processing is made possible by the very regular nature of image data (in particular, pixel neighbourhood is known) and dedicated powerful graphics hardware (Graphics Cards with dedicated in silico parallel processing pipelines).

Principle

Image sensing is an approximation of the complex nature of the light signal emitted/reflected by an object at a given time. A camera is only a receiving sensors, making it a passive method.

The camera has an array of photo-sensible sensors. Each sensor counts the number of photons arriving during a given time, thus gives an average intensity of light signal over a short time. Matrices of coloured filters allow acquiring the intensity of different parts of the light spectrum (e.g. (Red,Green,Blue) colours).

Aerial image

We differentiate aerial images (See Figure 14) from street images as the cameras are usually significantly different.

Aerial images have extensive geographic coverage because of the near uniform acquisition process and many satellites available. In urban context, they tend to have massive

Figure 14: Aerial image data.

occlusion because street canyons are occluding parts of the city. Several passes with different acquisition angles and directions can help reduce this problem (Garcia-Dorado and Aliaga, 2013).

Radiometric quality is generally high and distortion low. Thzse images are precisely geolocated (i.e. we precisely know from where and in which direction the image was taken relatively to known ground features).

Other spectrum than human-visible colours are often available and give precious information (e.g. near infra-red for tree detection). Pixel width is typically between 0.1 and 1 meter.

Street image

Street views (13) are usually taken from the ground by a person or a dedicated vehicle circulating the streets. These images allow seeing in great detail buildings and streets.

Geolocation of images is done trough GPS and inertia sensors, but a centimetric registration of those images is still an open problem.

It may be hard to coherently use multiple images.

Spatial coherence is difficult to obtain because of the registration challenge, temporal coherence because significant parts of the images may be occluded by moving objects, and radiometric coherence (colours) because the lighting conditions may change very quickly (e.g. moving from shadow to direct sun light).

The pixel size varies with the depth of the image but can be estimated from 1 to 10cm average, and the data volume is very sizeable (thousands of images per hour).

It is common to use multiple images to create sparse point clouds using a method called "structure from motion" (SfM). However such point-clouds are very different

from those obtained by Lidar. Due to instrinsic 3D reconstruction ambiguities, errors and noise are typically higher, and point sets are sparser in uniform area (e.g. a white texture-less wall). This differences partially explains why many methods are specific to Lidar pointcloud or SfM pointcloud. (Musialski, Wimmer, and Wonka, 2012, Sec. 2.2) give an introduction to SFM.

Raster data

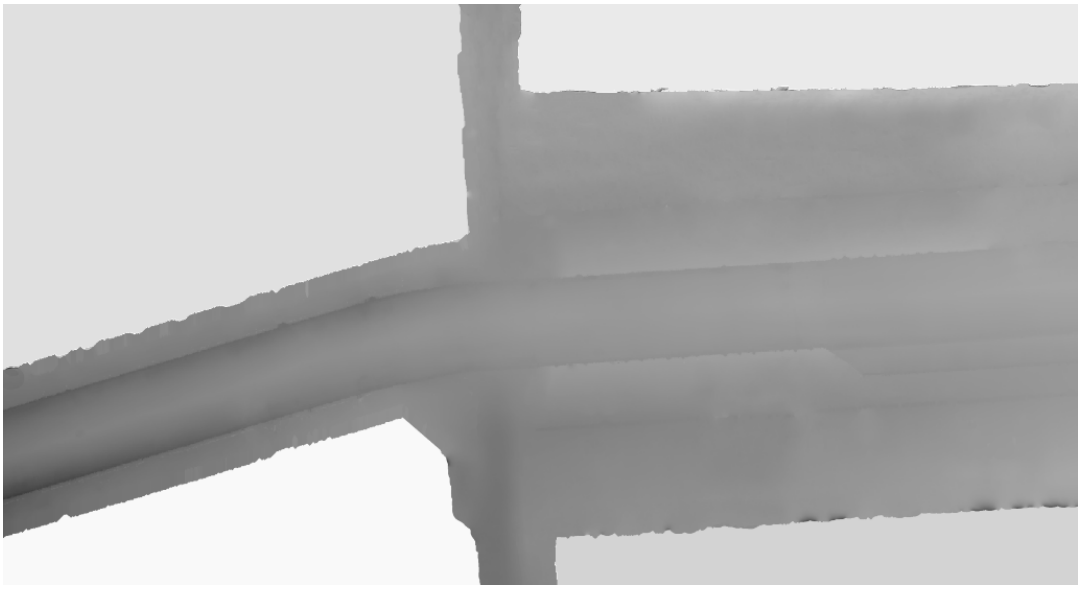


Figure 15: Raster Digital Terrain Model (DTM) data.

Introduction

In Geographical Information Science (GIS), a raster dataset is a regularly sampled 2D distribution (i.e an image) draped over a portion of ground (i.e viewed from above). This image can contain colors channel (RGB), but also any sampled field values.

The information contained can then simply be a visual texture (an ortho-photography, making it close to images (Section 1.4.2). But it can also be more abstract, giving for example some geometric information (the estimated height of roofs in a given area), or semantic information (the probability that the space the pixels cover are made of vegetation), or even statistical (an estimation of the average income distribution over a city). The Figure 15 shows a raster representing the ground elevation.

Level Of Details (LOD)

Rasters can be very large images (covering large ground with many small pixels). Thus they typically require a Level Of Details (LOD) approach. A raster can be tiled (regularly cut into smaller pieces) to access only a part of interest, and/or used at varied resolution (e.g. pyramid representation of the JPEG 2000 standard). Note that point cloud can also use LOD approach (See Appendix 7).

Details

Rasters are prone to quantization errors (intuitively, it is hard to represent a curve with rectangular pixels). They also have an obvious limitation: they can represent a 2.5D surface but not all 3D form or volume.

It is important to note that raster can be more semantically abstract than images and Lidar because they can sample any distribution. Thus the values can represent other things than direct sensing data (e.g. a land use map).

Vector data

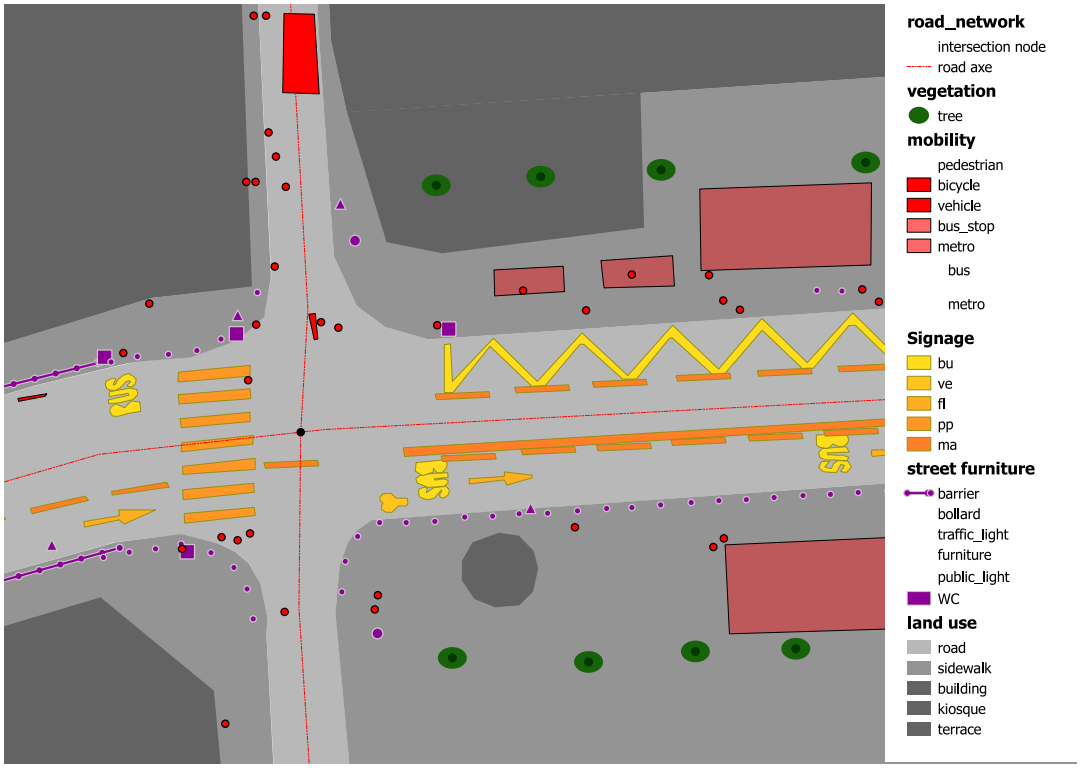


Figure 16: Vector data: various vector types forming a map.

Intro

Vector Data are classical data in map making (See Figure 16 which illustrates a number of vector data forming a map). Intuitively, vector data are arbitrary (mostly 2D) simple parametrized shapes with attributes. Usually the shapes are limited to points, poly-lines and polygons with holes. More generic forms like curve (arc of circles,Bezier curves, splines) , or 3D primitives (meshes, triangulated networks) are less common.
Attributes are values attached to a shape. For instance a tree could be represented as a polygon for the boundary of its trunk at ground height, along with the attribute ”tree_species” (e.g. "Platanus") and ”height_of_the_tree” (e.g. "12.4").

Abstract data

Vector data are usually more abstract than the other data by nature and by usage. By nature because both images and lidar point clouds can be represented losslessly with vectors. However vector data is irregular by nature (no information on neighbourhood). As a consequence representing images as grid of rectangle vectors is of reduced interest.

Vector data are also traditionally more abstract by usage. For instance a 2D polygon would be very classically used to represent a building footprint. In this case this polygon is already a simplified model of the building. Given the height, we could extrude the footprint to create a simple building volume.

Obtaining vectors

Vectors are not a direct result of sensing, but an interpretation of reality. Thus vector data can be obtained by an analysis of direct sensing data, which is a part of the objectives of the remote sensing research community (e.g road extraction (Bar Hillel et al., 2012)).

Vectors can also be man-drawn (using aerial images in background), produced by field survey (involving a positional device to map the objects), or extracted from pre-existing maps (vectorisation).

Vectors are not regular by nature (no information about neighbourhood), however using the attributes one can create a so called "topological model", which allows to create a graph structure over vectors. For instance a road axis network is composed of road axis (vector) with topological information (axis i and j are connected at node n).

A typical example would be that for each vector an attributes gives its connected vectors with some orientation information. This data structure is harder to manage and to use but allows minimizing data duplication. Using such a data structure enable different applications like traffic simulation, or advanced spatial-relationship analysis.

APPROACHES

In this section, we first propose a transverse classification of methods reconstructing/-modelling the different urban aspects. Then, we give pointers for procedural modelling, grammar, and inverse procedural modelling.

Transverse reconstruction method classification

Reconstruction strategies ordered by model importance

In the rest of this work, we consider a short state of the art for each aspect of urban modelling/reconstruction (building reconstruction (Sec. 1.6), street reconstruction (Sec. 1.7 on page 28), street network reconstruction (Sec. 1.8 on page 31), urban features reconstruction/modelling (vegetation (Sec. 1.9 on page 35), and urban objects reconstruction/modelling (Sec. 1.10 on page 40).

These aspects are very different in nature, type of data used, and type of results. Therefore we propose several classifications for the methods of each aspects. Each classification is intended as a way to compare methods.

All methods deal with the reconstruction of one aspect of urban model, we propose a transverse classification of these methods. We choose to classify reconstruction strategies

by the role the model play in the reconstruction method. At one end of the spectrum, the strategy of direct reconstruction from sensing data (e.g. triangulate a point cloud for instance). In this strategy, the model has a very small role, as it is mostly implicit.

At the other end of the spectrum, the strategy of catalogue matching. In this case the reconstruction strategy is to identify which model represents best the data, therefore, the model play a very large role.

This classification is illustrated in Figure 5 on page 9.

DATA-DRIVEN RECONSTRUCTION Some methods reconstruct directly from sensing data (low level reconstruction, data-driven), for instance reconstructing the ground surface, the building approximate geometry, the road network from image, etc. Similarly, points or pixels classification can be seen as low level reconstruction. These methods have the advantage to rely on an implicit model which may be very generic. Yet, the sensing data is often sparse and of relative low quality considering the scale of the considered objects. Moreover, low level reconstruction methods seems to be ill adapted to output structured/complex results (for instance a facade organisation, a hierarchical road network, a graph of parts of a man made object, etc.).

MODEL-DRIVEN RECONSTRUCTION A way to simplify a problem too wide is to add constraints and knowledge about it. Some of the methods therefore add strong hypothesis about the object to reconstruct, typically exploiting prior knowledge (road slope and turning radius is constrained by civil engineering rules, trees tends to grow to maximise exposition to sun light, etc.), and hypothesis of symmetries.

These prior knowledge are then expressed as strong models (Template, pattern, etc), and the reconstruction is much more model-driven (top-down). For instance when reconstructing road markings of pedestrian crossing, we can use the hypothesis that each strip is a rectangle, and that related strips are parallel with a regular spacing.

PROCEDURAL MODELLING However template and pattern become difficult to use when the reconstructed objects follow complex patterns and/or hierarchical patterns. In this case, procedural model offers a powerful and adaptable way to construct such results (for instance, expressing a tree procedurally).

When reconstructed objects have important and structuring relationship, a grammar is a good tool to formalise these while keeping a strong modelling power (for instance, using a facade grammar, shutters would necessarily be created and linked to a window). Moreover grammar are very hierarchical by nature, which suits well a number of aspects of urban reconstruction, as both natural and man-made object express .

INVERSE PROCEDURAL MODELLING Procedural modelling and grammar modelling have great modelling power, but are hard to use in reconstruction. Indeed, they can be used to create a model, but are hard to adapt to model something in particularr. In this case the paradigm of Inverse Procedural Modelling is necessary, that is given a model and observations of the object to be reconstructed, what are the parameters and rules of the model that best suits the observations (for instance, given a pedestrian crossing detection, what is its orientation, width, number of bands, etc.). The number of parameters to consider is extremely large, and this, in addition to sparse and noisy observations, may lead to an intractable problem.

CATALOGUE MATCHING In some case, the objects to reconstruct are very well known and may have very little variations. Thus, we can adopt a catalogue matching strategy. Instead of reconstructing an object, we use observations of the object to find the model that is the best match in a large model database. For instance, using a streetview we detect a urban furniture. The image is matched to a database of 3D model of street furniture. The best model is then scaled and oriented. Please note that in this case, the model almost totally determine the result. This allows to decompose the reconstruction problem: First find which street object is where, possibly determining some of its properties, such as its orientation. Then, find or generate a similar 3D model and populate the reconstructed street with it.

Additional considerations for reconstruction strategies

INTERACTION Independently of the strategy used to reconstruct object, a user interaction is often necessary. This is especially the case when input data can not be really trusted, or when the reconstruction method strongly relies on model (procedural modelling for instance). Controlling grammar is difficult and dedicated methods may have to be tailored (for instance, using brush to describe the different parts of a city; or a street network may be generated basing its morphology onto the surroundings).

UPDATING DATABASE AND FUSION Most of the methods we presented are straightforward modelling/reconstruction methods working on sensing data. However, for real life application (especially street network reconstruction), one may use not only sensing data, but also a previous coarser results. For instance an incomplete road network is completed with road extracted from sensing data. These methods are still about reconstruction, but they may also contains supplementary parts such as data fusion, data qualification, etc.

Procedural modelling and grammar

We recommend the read of the seminal article of Müller et al., 2006; Parish and Müller, 2001 for an introduction to shape grammar (The Figure 17 is extracted from Parish and Müller, 2001).

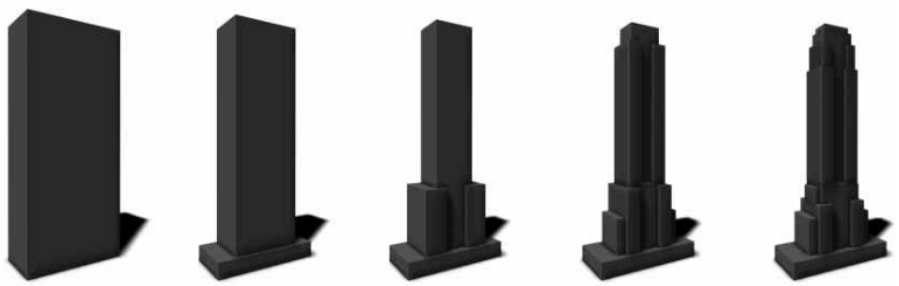


Figure 17: An example of usage of shape grammar from the seminal article of Parish and Müller, 2001.

Inverse procedural modelling

Inverse procedural modelling is the paradigm where a procedural model is fitted to observations. It is important to note that we do not only look for the parameters of the model, but also for the rules used in the model (i.e. the number of parameters is not fixed). For instance in the case of a facade grammar, we do not only look for the number of floors, but for the rules that will be used to generate these floors (for instance, create window with balcony and shutter).

BUILDINGS AND FAÇADES

Building reconstruction has received much attention in the past decade. Thus, methods have focused on diverse parts of buildings reconstruction, (facade reconstruction, roof reconstruction, indoor reconstruction, etc.).

Different types of building may also be reconstructed using different methods (Manhattan/Atlanta/Planar-hinged building type (Garcia-Dorado and Aliaga, 2013) or suburban house (Lin et al., 2013)). Some methods focus on large scale solution, efficient visualization, Level Of Detail feature, etc.

The methods used are so diverse that the author of the recent state of the art (Musialski, Wimmer, and Wonka, 2012) have chosen a straight order by goal and data input. Klavdianos, Zhang, and Izquierdo, 2013 also establish a building reconstruction state of the art.

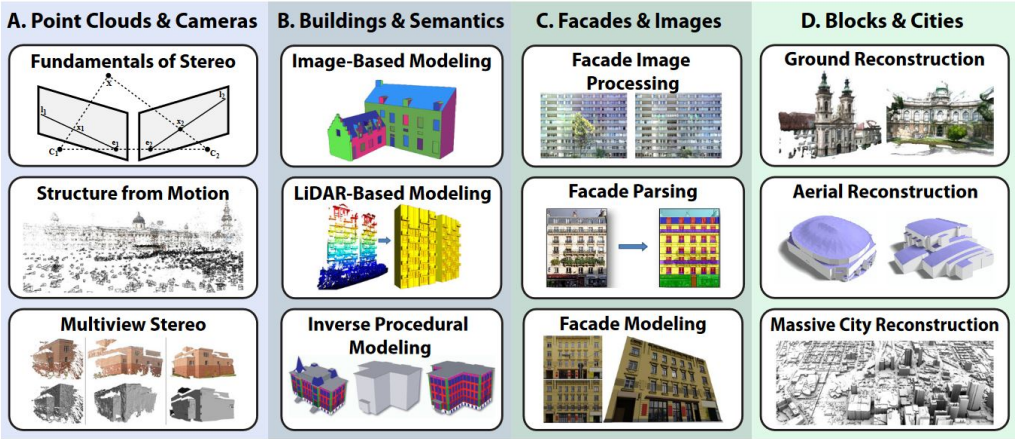


Figure 18: Illustration of Musialski, Wimmer, and Wonka, 2012 illustrating building reconstruction.

We refer to the Figure 18 extracted from (Musialski, Wimmer, and Wonka, 2012) for a quick overview of different approaches for building reconstruction.

We strongly recommend to read both these states of the arts articles for building reconstruction, as many of the strategies explained could be used for the reconstruction of other objects. We also feel that the building reconstruction community has pioneered many advanced articles about shape grammar and inverse procedural modelling.

RC:1.7.0.0. chap1: street : add illustration !

Introduction to street reconstruction

Challenges for street reconstruction

Streets are essential components of a city model. As the medium pervading all other structures and objects they are complex. First the geometric nature of streets is specific, detailed and not normalized.
Second, a street is a complex arrangement of objects that are inter-related and have their own structure.
Third, streets are functional objects used by the inhabitants of the city for their displacement.

Thus, a street organisation is partly guided by these functions, and as such, street reconstruction should provide an ouput compatible with this functions.

STREETS ARE COMPLEX, EVEN FOR HUMAN Streets are so familiar places that we specialise very early in using them during childhood. However one can remember the complexity of the task when travelling in another country. There, every aspect of a street can be different.

Children have to be taught a long time where and when to walk, not speaking about driving rules, or using the public transportation system, which are even more complex tasks. In the spirit, people with even a light Intellectual or physical disability may have significant trouble navigating the public transportation system.

STREET FOR TRAFFIC An essential function of street is multi modal navigation (vehicle, public transport, bikes and pedestrian). Such navigation uses network level features (Section 1.8) which have great impact at the street level organisation. For instance the sole purpose of street markings is to support traffics. Being on the ground they are prone to occlusion and wear, but there use is strictly regulated (e.g. France reference document is 65 pages long, (French Ministry, 2012)).

Considering that streets are used for several transportation methods (pedestrian, bike, public transport, vehicle) that are mixed (e.g. a pedestrian crossing is shared between pedestrians and vehicles), reconstruction street is difficult.

STREETS ARE ORGANISED Streets are challenging to model because they form a partially organized structure (typically organised relatively to the central axis), yet are much less locally regular than a building (in particular, the relations are more generalised, like intrinsic partial symmetry vs extrinsic, see (Mitra et al., 2012), ch. 7). Also, the street components have strong relations between them, which makes difficult to model a small area at time (in opposition to building which can reasonably be defined as dissociated from the other close buildings). For example, reconstructing a pedestrian crossing usually implies there would be another in the next hundreds of meters. Some street features may follow a partial symmetry (bollards for instance), a pattern

(pedestrian crossing markings bands), or be organised in inter-related hierarchy (lane markings and traffic light) .

STREETS ARE HARD TO SENSE Lastly the data collection is difficult. Aerial sensing may be impeded by buildings, and the street geometry and features make it difficult to avoid large occlusions due to traffic, people, trees... In opposition to building, whose main feature (dor, windows, etc) are large (1m), street are partially organised by kerb (separator between road and sidewalk) which are much smaller (0.1m). The necessary geometric precision is even greater when considering slope and water drainage.

RELATED WORK There are been remarkably few works on reconstructing streets, even if streets contribution to a city model is evident. We can conjecture that this is partly due to the fact that data to the required precision (less than 0.1m for a basic curb) has been only recently available in urban environment. Also, the geometric nature (streets are not necessary blocky), the diverse and complex arrangement of objects (markings, signs, furniture), and the dependency on many research fields (object detection & segmentation, pattern recognition, paving and texturing, intrinsic symmetry detection) makes the problem challenging.

However the simulation industry has used dedicated data models (for instance, RoadXML[1] or OpenDrive[2]), which characteristically include network aspect, surface material and 3D representation, along with road objects and road related objects. Powerful specialised softwares allow to design intersections in all their aspects (lane size & position, traffic regulation depending on the traffic throughput, regulatory material) in the construction and CAD field. These software are not included in this state of the art, as we were not able to test them.

Street are also part of a street network, this constraint needs to be enforced at all time, but can provide precious information (e.g. traffic direction(s), thus orientation of traffic signs, etc.). Therefore some methods for street network modelling can also be partially applied to model streets. Even if it has been a common practice, modelling streets like the complimentary space of buildings is not sufficient (Cornelis et al., 2008) for many applications, and in many cases simply erroneous (e.g. private garden, places, parks).

Modelling the geometry of the street

One should model the detailed geometry of the street and curb (which is typically varying to separate pedestrian crossings or driveway entrances). This is counter-intuitively difficult. Such process cannot rely on Manhattan-like hypothesis, and must deal with the precision issue.

Some road model from road network modelling methods can be applied (See Section 1.8 on page 31) For instance road geometry can be modelled as 3D clothoid (Applegate, Laycock, and Day, 2011; Bertails-Descoubes, 2012; Galin et al., 2010; McCrae and Singh, 2009b), arc and line pieces (Wilkie et al., 2012), polynomial model (Hervieu and Soheilian, 2013), B-Spline ((local only) (Wedel et al., 2009)), using road profile (Despine and Baillard, 2011), or using brute mesh (Cabral et al., 2009). The reconstruction of the geometry of the street can also be made at a more general level by sampling height

1 www.road-xml.org
2 www.opendrive.org

(Digital Elevation Map from aerial imaging or Lidar, ground Lidar (Kumar et al., 2013), detecting the limit of vehicle track)). However reaching the required precision might be difficult. In fact, even with massive terrestrial data, automatically dealing with occlusion to get a coherent street geometry is still an open problem (Hervieu and Soheilian, 2013; Serna and Marcotegui, 2013).

Object detection, primitive extraction

But street are also a subtle arrangement of related objects. Street objects (like vegetation (Section 1.9), street furnitures and markings (Section 1.10) are hard to deal with individually. Detection is already a hard problem, reconstruction is even harder. Detecting objects in street is challenging due to variety of objects and occlusion. (Golovinskiy, Kim, and Funkhouser, 2009) use a four steps method to detect objects in street Lidar: localisation of objects, segmentation, feature extraction and classification for a small amount of objects. Beside comprehensive testing and proposing several alternatives regarding classification methods, they also reach the same conclusion about the importance of relations between objects and use ad-hoc features to this end ("contextual feature"). (Zhang, Wang, and Yang, 2010) demonstrate the possibility to perform urban segmentation based on depth map extracted from video. Local features are extracted from depth map (height, planarity, distance to camera), then a random forest classifier followed by a graph cut minimization methods output a labelled segmentation. (Yu et al., 2011b) focus more on segmentation with a basic classification, but their method could be used as primary step for detailed classification. Similarly, (Lafarge et al., 2013) automatically extract primitives (e.g. plan cylinder, torus, etc.) from point cloud obtained by SfM. Their goal is more toward mesh compression and partial holes filling, but such primitives could also be used for object segmentation. Although the core of their method is a planar-based residential house reconstruction, (Lin et al., 2013) also detect objects (mailbox, plant, road sign, streetlight, waste bin) using an adapted version of (Zhang, Wang, and Yang, 2010).

Whatever the method, the number of type of object detected is small (about 10) and the error rate varies a lot depending on type of objects. The research field of façade reconstruction had the same type problem. The trend to resolve it has been toward leveraging the organisation and relations of objects (contextual information).

Therefore we include in this state of the art a prospective consideration of street related object relation detection and analysis.

Relation between objects

At the street level it is possible to leverage the pattern and inter-relation of this object to gain critical information about objects. The Figure 8 on page 15 clearly shows that street objects are strongly organised (top), to the point where removing this organisation (bottom right) makes no sense. Defining and retrieving relations amongst objects is an old and multidisciplinary problem. (Clementini and Laurini, 2008) review related references in linguistic, philosophy, psychology, Geographical Information System (GIS), Image processing and qualitative spatial reasoning. They propose a common evaluation framework.

More related to the GIS community, (Steiniger and Weibel, 2007) present a coherent typology of spatial relations applied to cartographic generalisation. Their typology is general enough to be applied outside of this field. In a recent state of the art, (Touya et al., 2014) describe in great details previous works in the GIS field and propose a new taxonomy along with several use cases to illustrate the relations.

Extracting such relations is a difficult problem, and could be related to Extrinsic/intrinsic symmetries (Mitra et al., 2012). The real world relations are fuzzy like in the method used by (Vanegas, Bloch, and Inglada, 2013) to extract almost exact fuzzy spatial relationship in aerial images. More generally, complex pattern of objects may need a full grammar to be represented (See Section 1.5.2 on page 26).

Texture synthesis

When the objective is to get a photo-realist 3D model, a possible strategy is to use real or synthetic images and drap them on a geometric street model. This texturing process (or draping) is a major bottleneck for reconstructing a large number of streets. Such textures are hard to design, and if using data from sensing, they have to be cleaned. (Cornelis et al., 2008) uses multi-images to blur the detected vehicles and replace them by detailed 3D model. They also use texture map to efficiently store the road and buildings aspects. A state of the art of texture synthesis and deformation is out of the scope of this article (interested reader could refer to (Wei et al., 2009)). To pick a few, (Cabral et al., 2009) uses generic texture deformation by auto-similarity maps to adapt to the geometric deformation. Also, although the focus is not the same, (Ijiri et al., 2008) could be used to generate sometime complex pavement pattern of streets ground.

Conclusion about street reconstruction

Street reconstruction is a difficult problem, which is essential for urban reconstruction, but seems to have been much less studied that building reconstruction. Related works focus on road (not street), and usually only reconstruct one aspect of a street (geometry, transport related information, street feature, etc.).

We note that street functions (transportation) and features (objects) are closely interrelated, which indicates the need of a global method taking both into account.
It seems that streets are strongly determined by their transportation function. As such, the role each street plays in the more global street network is a key factor that has to be taken into account when reconstructing this street. This indicates the necessity to have a multi-scale approach, both at street and street network scale.
Street objects have complex organisation (pattern, symmetry), are interrelated, and may also depend on street morphology. This indicates that a very powerful approach able to mode hierarchical patterns is needed. Procedural and/or grammar approach appear to be good candidates for this task.

STREET NETWORK

RC:1.8.0.0. chap1:street network : add illustrations

In this section we introduce challenges and stakes of street network reconstruction, then propose three classifications of street network reconstruction methods. The first classification is by the type of road network that is outputted. The second classification is by the type of input used. The last classification is by the type of road model used.

Overall, the type of road network output range from simple network to hierarchical network to fully attributed network for traffic simulation. The input can be from example/template, procedurally (without or with interface), specifying constraints or using GIS data.

Popular procedural methods are L-system, Agent-based simulation, and templates. ((Kelly and McCabe, 2006), page 12).

Introduction to street network reconstruction

Street network modelling is of particular importance for city modelling. A city organisation relies so heavily on street network that it is often the first step of the city modelling process(e.g. CityEngine (ESRI, n.d.)). Street are also connected and form a network regulated by traffic laws and many related signs, thus having a specific nature which must be taken into account to enable traffic simulation. The street network then becomes a complex graph which exhibit a partial fractal nature (Frankhauser, 2008). Reconstructing becomes then much more difficult because the support as well as the connectivity information must be retrieved and coherent.

Challenges in street network reconstruction

In urban planning designing the properties of the road network is essential for the city growth and for a good interfacing with city surroundings. The financial and environment-related impacts are also enormous (e.g. road network is commonly used to open up neighbourhood, which can significantly increase land price. On the opposite a major urban road can negatively separate a neighbourhood into disconnected pieces, thus weakening the urban fabric). The street network is also the support of several forms of transport which are entangled. This fact has important repercussions on city reconstruction. A street with major vehicle traffic and bus lanes will be morphologically and functionally very different from a pedestrian street.

Moreover, reconstructing a city without street network would be pointless because the street network is the very object that links every others and assure the connectivity of the urban fabric.

Why reconstruct street network

Reconstructing the street network is essential for numerous applications, being for direct use (traffic simulation), or for indirect information (e.g. gives complementary information about a street that could be used for street morphological properties evaluation for realism or environmental simulation). Real world road network maintenance and construction is a massive industry (around 0.5% of GDP in Europe, according to (European Union Road Federation, 2012, pages 29-30).

At such it is not surprising that major CAD software companies like Bentley[3] and Autodesk[4] propose advanced products to create/renovate road networks. These software features would probably place them at the state of the art, however due to lack of related publications we can not discuss them furthermore.

We note than for procedural city modelling, constructing the street network is often the first task ((Parish and Müller, 2001) and following), because street structures the city.

Street network and road network

Most of the methods we consider reconstruct road network, and not necessary road network in urban environment, even less street network.

As such, these methods focus on reconstructing a network for vehicle, although streets contains other network, such as pedestrian network. Yet pedestrian network can be inferred from road and building (**Ballester2011**), or semi-automatically created with ad-hoc tool (Yirci et al., 2013) and then updated afterward using GPS trajectories (**Park2015**).

We added methods performing road network reconstruction to this state of the art as they may be applied for streets.

A classification of street network reconstruction methods

In this section we propose three classification of street network reconstruction methods: by targeted road network model complexity, by Input type and by road type.

Classification by targeted road network model complexity

We propose a first ordering of related article by the type of road network they output.

FLAT ROAD NETWORK Some methods are suited to design flat road network (Applegate, Laycock, and Day, 2011; Galin et al., 2010; McCrae and Singh, 2009b; Merrell and Manocha, 2011). These roads may adapt to terrain geometry and/or constraint (lake, slope, forbidden area).

HIERARCHICAL ROAD NETWORK Yet road network is intrinsically hierarchical (motorway, primary way, etc.), procedural methods are particularly adapted for this. For instance, (Chen et al., 2008; Galin et al., 2011; Lipp et al., 2011; Parish and Müller, 2001; Yang et al., 2013) use multi scale methods, but usually only consider the graphical aspect.

ROAD NETWORK WITH TRAFFIC INFORMATION Lastly methods can output a complete road network with full navigation attributes for traffic simulation and/or visualisation (Despine and Baillard, 2011; Wilkie et al., 2012). These methods are more focused on filtering, correcting errors, constructing a multi-layer data model (global topological network, lane network for traffic, geometry+texture for visualisation).

3 www.bentley.com/
4 www.autodesk.com/

Classification by Input type

Methods about street network reconstruction can also be ordered given their data input type.

First some methods directly reconstruct road network using results from sensing, such as aerial Lidar (Wang and Weng, 2013), aerial image and radar Chu He et al., 2013, GPS trace (Ahmed et al., 2014; Kuntzsch, Sester, and Brenner, 2015), or even mobile mapping (Mueller et al., 2011).

(Merrell and Manocha, 2011) is example-based (a given model is analysed, then extrapolated to bigger model). A template also plays a role in (Parish and Müller, 2001) to determine the global pattern of road configuration (e.g. dominant grid-pattern as Manhattan, or dominant radial pattern as Paris). Similarly, templates are used for high level road network configuration in (Yang et al., 2013), but more importantly to design minor roads (indirectly).

The principal weakness of procedural generation is control (See Section 1.5.2). Thus many methods try to deal with this by providing interfaces. In (Applegate, Laycock, and Day, 2011; McCrae and Singh, 2009b) user directly sketches road path in 2D and a 3D clothoid is fitted to the correct elevation and the land is properly dug. In (Lipp et al., 2011) a user directly edits the network graph with advanced operations (copy-past, insertion, rotation, translation) that preserve the graph properties. (Yang et al., 2013) propose some control via constraint layers (e.g. a lake surface, or a given type of organisation for an area).

Similarly, many methods use constraints as input. Typically mechanisms permit to define area where road network is constrained, for instance in parks and/or river. (Chen et al., 2008; Lipp et al., 2011; Parish and Müller, 2001).

It is different for (Galin et al., 2010, 2011) where the constraints system is at the hearth of the method. In these articles, custom cost functions, special constrains (park, highway without intersection) and a specialised solving system allow the system to generate an optimal path for the road taking into account the geometry and the nature of the terrain (constructing bridges or tunnels along the way).

Input data can be even more abstract as in (Despine and Baillard, 2011; Wilkie et al., 2012), where they use polylines with attributes. The challenges are then as much to filter and correct input as to use methods to generate a complete road network data suitable for traffic simulation.

Classification by road types

We can also classify the methods for street network reconstruction by the way they model the road surface.

The clothoid is a popular way to model road. This is due to the fact that clothoids are mathematical curves along which curvature varies linearly, thus conducting to a pleasant acceleration while driving. Interestingly this property is actually used by road engineering software. Clothoid can be extended to piece-wise clothoid or super-clothoid (Bertails-Descoubes, 2012). In urban environment, acceleration constraints are often less important that historical heritage or global city layout, thus the model iseems to be less used.

Another popular parametric model is based on arcs (circular arc: (Wilkie et al., 2012), parabolic arcs: (Despine and Baillard, 2011), or just polylines: (Parish and Müller, 2001)). See (Wilkie et al., 2012, pages 2-3) for more geometric primitives for road modelling.

Other potential methods

We detected two methods that are not directly related to road network modelling but may be used to this end.

For instance (Merrell and Manocha, 2011) is a very general procedural modelling method that analyse an input shape (geometric constraint) in order to create a new bigger procedural model, respecting some user defined constraints. In a different direction, (Krecklau and Kobbelt, 2011) propose a custom grammar adapted to interconnected structures. By defining potential attachment points, and geometrical queries able to find potential connections, there grammar allows to model different kind of interconnections. This may be naturally extended to road network modelling, taking advantage of the connectivity that defines a road network.

Conclusion

The street network is essential for urban reconstruction, as it defines many aspects of the city, and is paramount in the way streets are used. Even more important, the street network is a structuring element for a city, similarly to how the street axis is structuring for street. This indicates that an urban model could be based upon the street network.

Most methods focus on road network, few consider urban environment, and no method reconstruct a real street network, including pedestrian network, and vehicle network. In the same spirit, not all method produce a hierarchical network, even fewer with geometry and traffic information.

The difficulty seems to be coming from the fact that a streets network is a graph embedded in 3D, which makes it much more abstract than the sensing information, hence the complexity.

Because the problem is complex, the integration of user input is necessary in many methods. This indicates that an interaction with the street network is important and necessary.

Some aspects of street network are impossible to determine without street features. For instance the number of lanes has to be inferred from markings, the connectivity of the network from traffic lights and traffic signs, etc. This seems to indicate that a street network reconstruction has to be done at two scales: at the network scale and at the street scale.

URBAN VEGETATION RECONSTRUCTION

RC:1.9.0.0. chap1: vegetation : put illustration

In this section, we introduce why reconstructing urban vegetation is an important part of urban modelling, and which challenges it creates. We then discuss vegetation reconstruction and the various strategies and scale at which it can be done, then we propose three classifications of vegetation reconstruction methods.

Introduction

Why reconstruct vegetation in urban area

VEGETATION PLAYS AN IMPORTANT ROLE FOR CITY The vegetation has been primordial for Mankind for a long time. Forests occupy a large part of land surface (30% in France). It is then not a surprise that the vegetation is very common and plays a very important role in cities.

The vegetation in cities has a significant influence on noise propagation, air quality and temperature, water cycle, and also has a significant impact on human social behaviour. Each of these aspects covers a vital part of urban planning, be it for comfort (temperature, air quality, human behaviour), or for technical advantages management advantages (water cycle, noise, wind).

VEGETATION PLAY AN IMPORTANT PART FOR CITY MODELLING In a pure 3D reconstruction, the vegetation is important for realism and because it is geometrically so different from its surrounding (a sparse organic spherical form, as opposed to the locally planar and compact rectangular form of buildings or streets). As such, methods devised for buildings reconstruction are usually sub-optimal at best for tree modelling. Because trees are large and recognisable, they alter much the perception of a street. Moreover, modelling of vegetation is important because vegetation has such a strong role for cities.

VEGETATION IS VERY PRESENT IN CITY As a numerical example, about 5% of Paris surface is dedicated to parks, that is not taking into account the two small forest that are officially within Paris (bois de Vincenne and Bois de Boulogne). The number of trees in streets is above 250 000 in Paris. This means that in average there are trees every few dozen meters in Paris streets.

VEGETATION RECONSTRUCTION IS USEFUL FOR OTHER METHODS Even when reconstruction of vegetation is not explicitly wanted, it can be of great help to have a vegetation model (possibly implicit) for reconstructing other objects occluded by vegetation. Another important use of vegetation is for landmark maps. In such context the large visual space a tree occupies is precious because it is easily recognised.(See (Soheilian et al., 2013) for a state of the art of landmark based localisation, and (Brenner, 2010) for a localisation using exclusively trees).

Challenges in urban vegetation reconstruction

Reconstructing the vegetation in an urban environment is challenging for several reasons, some due to the nature of the vegetation (multiscale, ecosystem), some more technical (sensing data precision and completeness, scaling).

VEGETATION IS A MULTI-SCALE COMPLEX ECOSYSTEM Vegetation is often a whole ecosystem, with several species living together. Like many living organisms, plants exhibit a fascinating multi-scale nature with fractal-like properties.

Therefore one must define up to which scale the reconstruction process should stop.

To the best of our knowledge the current state of the art for trees is at the branch scale, with reconstructed trees having a similar leaf organisation as the model (Pirk et al., 2012), but not an exact leaf to leaf reconstruction. However a recent work on small plants suggests a future move toward the leaf scale (Li et al., 2013). Concerning the vegetation reconstruction, most of the works reconstruct the vegetation in the form of a distribution of species.

THE LARGE NUMBER AND SCALING CHALLENGE The vegetation uses large amount of city surface, and in streets each tree may occupy a large volume. Moreover, the scaling problem is evident when considering that each tree may have hundred of branches, and there are hundred of thousands of trees.

MULTI SCALE REPRESENTATION OF TREES FOR FAST REAL TIME RENDERING As stated into the introduction, trees present a multi-scale fractal nature. This can be typically leveraged to allow the efficient modelling of large areas with many trees. A less detailed model can be rendered when the tree is far from the viewer, while the more precise model is showed when the tree is close. (Livny et al., 2011) produce different levels of details for every model of trees. Similarly, the popular XFrog[5] can also be used to produce levels of details. When the trees are regrouped, one could also rely on tailored methods to allow realistic and fast visualisation (e.g. (Bruneton and Neyret, 2012)).

VEGETATION IS HARD TO SENSE Another point is that trees are by nature occluding elements from an aerial point of view. This stems from the tendency of the trees to capture sun light coming from above, hence they limiting the picturing. For this reason, removing the trees for correct façade reconstruction is a very classical problem in terrestrial laser and image processing.

Tree reconstruction is also challenging because the sensors (image, Lidar) give information about surfaces, which is fine for a building, but may fail to pass the tree crown to get the branching structure (full wave or hyperspectral Lidar somehow mitigates this).

Vegetation reconstruction

A global state of the art on vegetation modelling and reconstruction is out of scope of this work; therefore we will only give an overview of vegetation modelling and focus on its use in urban context. We also included some method for tree modelling, as these could potentially be used for tree reconstruction using an Inverse Procedural Modelling paradigm.

In this section, we briefly introduce the strategies for vegetation reconstrution, then consider the different scales to which the vegetation can be reconstructed. We then propose three classifications of methods for vegetation reconstruction.

Strategies for vegetation reconstruction

FOCUSSED ON TREES Vegetation reconstruction usually focuses on tree reconstruction, even though some methods output an ecosystem type rather than a tree species

5 http://xfrog.com

(Gong, 2002). Orthogonally new Lidar technologies allow accessing more tree properties. For instance (Hakala et al., 2012; Wallace, Nichol, and Woodhouse, 2012) recover tree properties and canopy properties using multi spectrum Lidar technology.

A MORE MODEL ORIENTED RECONSTRUCTION The strategies for vegetation reconstruction are slightly different from typical strategies for man-made objects reconstruction (including buildings and façades). This is due to the fact that tree species evolve slowly and have been known for centuries, along with key properties of each species. Moreover, urban tree species are much more limited (order of magnitude : 100) than potential street furniture types for instance (order of magnitude : 10000).

For these reasons, and because of the multi-scale problem, top down approaches (model oriented) seems to be much more popular than bottom up approach (data oriented). That is, most model have strong hypothesis and model which are fitted to sensing, rather than directly using sensing to reconstruct trees from scratch.

This is quite different from building reconstruction, where building styles can be mixed, and each building does not necessarily fully enforces a style.

TREE RECONSTRUCTION OR TREE GROWING There are two main approaches to reconstruct the vegetation in cities: an analytical approach, where we try to retrieve direct morphological information about the tree to reconstruct it as is, and a more synthetic approach where we try to retrieve general information about the tree (species, height, crown seize), then synthesize it using growth model and known parameters of the species.

Choosing a scale for reconstruction

Vegetation is multiscale, therefore, before reconstructing, the targeted level of detail has to be chosen.

FOREST Forest management is a century old tradition. Therefore forest models have been developed, such as group of trees species repartition, possibly with their age, height, crown size, etc. These are used for forest exploitation, land planning and so. Such models are commonly obtained by field surveys, along with information obtained from remote sensing technologies (aerial images, Lidar) (Gong, 2002). For example (Watt et al., 2013) use full-wave ground Lidar to estimate two exploitation-related characteristics of a patch of forest.

PATCH OF TREES It is also common to model patches of ecosystem, with a larger scope than tree alone, sometimes involving plants modelling. This allows height/species/spatial statistical distribution analysis.

INDIVIDUAL TREE Tree models have been actively researched, including tree growth characteristics and species specificities. Procedural modelling methods are especially popular.

INDIVIDUAL PLANT Plant modelling is also an age old tradition (Van Gogh, 1888), with many applications in design and entertainment. More recently plant reconstruction has also been tackled (Li et al., 2013).

Classifications of urban vegetation reconstruction methods

We propose three classification of methods related to vegetation reconstruction.

Classification by input data type

We classify the vegetation reconstruction method based on the input they use, from dedicated Lidar to more generic remote sensing, to interactive feedback (human interaction).

Input data for tree modeling can be point clouds from Lidar tailored acquisition (Livny et al., 2011; Preuksakarn et al., 2010) or general acquisition (Livny et al., 2010), as well as point clouds from stereo (Li et al., 2013). Some methods also use aerial images (Iovan et al., 2013), or semantic maps (Beneš et al., 2011). Some methods are based on constraints on the tree growth (Pirk et al., 2012; Runions, Lane, and Prusinkiewicz, 2007; Talton et al., 2011). Lastly, many methods rely on user feedback but may be automated by unsing remote sensing data inputs (Krecklau, Manthei, and Kobbelt, 2012; Krecklau, Pavic, and Kobbelt, 2010; Lintermann and Deussen, 1999).

Classification by modelling method

We propose another classification of tree reconstruction following the modelling method they use, from procedural methods to L system to generic grammars.

Individual tree modeling is a mature research interest. It has been historically focused on procedural methods. Mature interactive commercial solutions such as XFrog (Lintermann and Deussen, 1999) exist and are widespread. In most cases the trees are modelled procedurally, possibly using parametrised shapes like generalized cylinders (Bloomenthal, 1985; Li et al., 2013; Pirk et al., 2012; Preuksakarn et al., 2010; Xfrog, 2014).

Explicit grammatical systems are also popular, in particular the L-System grammar (Deussen et al., 1998). More general grammars have been extended to produce trees along with more rectangular objects (Krecklau, Manthei, and Kobbelt, 2012; Krecklau, Pavic, and Kobbelt, 2010). See Section 1.5.2 for more details about procedural modelling.

Classification by Reconstruction strategy

The last classification of the reconstruction methods we propose is by reconstruction strategy, from direct from data, to analyse-synthesis to inverse procedural modelling to whole urban ecosystem design.

Reconstruction strategies can be straightforward (Livny et al., 2010; Preuksakarn et al., 2010) from direct remote sensing data. It requires however high quality data and has not been experimented on city scale. However the reconstructed trees can have similar look and properties as the real one up to the level of group of leafs (Livny et al., 2011), or even the leaf level (Li et al., 2013).

Other methods focus on an analysis-synthesis approach. The goal is to retrieve a number of properties of the tree (species, height), along with constraints introduced by its surrounding, then use a realistic growth method to obtain a tree model hopefully close to the real tree. (Runions, Lane, and Prusinkiewicz, 2007) use a space constraint approach to model the competition for space, while (Talton et al., 2011) constrain the tree leaf coverage by a bitmask, and (Pirk et al., 2012) add solid object constraints as

well as shadow influence. (Iovan et al., 2013) use images to detect and classify urban trees, then use the extracted parameters as well as space constraints to grow plausible urban trees.

(Beneš et al., 2011) are even more generic and introduce man-related constraints on a city area : in some part of the city vegetation growth is strictly controlled (trees species and spatial repartition), in other the control is less strict. Trees are also spreading over time. The system is then evolved over a period of time to generate 4D tree repartition and visualisation.

Conclusion for urban vegetation reconstruction

Vegetation is important for city modelling, both by its sheer presence, the roles it plays (temperature, pollution, noise, water, human perception, etc.), and its interest for urban modelling (street morphology, occlusion, landmark for registration).
Yet, the vegetation is hard to reconstruct (complexity,multi-scale, volume), and most methods focus on trees.

Because vegetation exhibit a regular and hierarchical nature, procedural modelling methods seem to be very indicated.

We note that the vegetation strongly depends on other urban features. Plant species will be influenced by the typology of area (residential, industrial, etc.), plant growing will be influenced by buildings, and realistic trees will most likely be pruned, therefore being influenced by road surface, and some road feature (road surface, traffic light, traffic sign).

URBAN FEATURES

RC:1.10.0.0. chap1 : urban feature : add illustration

introduction to urban feature reconstruction

We consider only man-made urban feature reconstruction (See Section 1.9 on page 35 for vegetation). We found few methods dedicated to urban feature reconstruction (street furniture, markings, etc.). Therefore we also integrate generic methods for man-made object reconstruction in this state of the art. We consider that these methods could also be applied on street objects.

Importance of street features for city

A city contains large amounts of street features, such a street furnitures, markings, etc. These are important by their number (over 1 million in Paris), by their diversity (over 13000 references on a site like (ArchiExpo, 2014)), and above all by the functions they fulfil (information, security, decoration, etc.). Street furnitures are seldom randomly placed and chosen, but instead are essential tools for the complex social interactions that a city host. Figure 8 on page 15 shows well how position and relations are important for urban features.

Virtually any human behaviour in a city relies on street objects, essentially because street objects regulate transport (information, rules, isolation, whatever the modality) and play a role into managing the city (waste, water collection).

Importance of urban feature modelling

Modelling street objects is then essential for traffic simulation, and also for realism (some piece of street furniture have achieved a landmark status, like Curitiba bus stations[6] in Brazil). Street furnitures can also be extracted to form a landmark map, thus assisting in the georeferencing of a vehicle or user with basic sensors ((Hofmann and Brenner, 2009)).

Challenges for urban feature reconstruction

Reconstructing urban features is difficult because of their relatively small size, essentially disabling any air sensing, and making it difficult to have precise and complete data (e.g., only a part of a parking meter would be on a street view or on a Lidar acquisition). The geometrical complexity may be high or deceptively simple (e.g. traffic signs are almost pure 2D). The material used can also complicate data sensing (glass, shiny metal, reflective paint). However such man-made objects typically expose strong regularities, symmetries, as well as a dominant plan-based structure which can be used by methods to improve reconstruction.

Reconstructing urban features

As always in a reconstruction problem, we have to define up to which scale the objects are to be reconstructed. For instance when reconstructing a street bench, shall we simply reconstruct the bench type and orientation, or shall we reconstruct it as several plans with texture, or shall we reconstruct each plank composing it, or shall we even reconstruct how the plank were bolted together, etc.

It seems that this level of reconstruction is dictated by the quantity and precision of input data, as well as how much the method is model driven. This problem is especially pregnant in streets, were the most precise data (order of magnitude of 0.01 m) are limitating, as well as the large occlusions.

Of course this level of reconstruction also depends on the intended applications, a proper generalisation is often necessary for performance reasons (trying to render the nails in the hundred of thousands of Paris street furnitures would most likely fail and be useless).

Input types

Traditionally street feature reconstruction methods use street lidar and images (Golovinskiy, Kim, and Funkhouser, 2009; Soheilian et al., 2013). In the more general object reconstruction field, other methods use noisy point cloud from images or color and depths devices (RGBD camera, like the Kinect) ((Stuckler, Biresev, and Behnke, 2012)). Even farther, some methods directly use 3D models ((Shapira et al., 2009)) to analyse structure and match it against a database. Some methods inputs are even more abstract, like a

6 https://en.wikipedia.org/wiki/Rede_Integrada_de_Transporte

set of relations among objects ((Yeh et al., 2012)), or interactive user inputs ((Gal et al., 2009)).

Hypothesis on street features

Street objects will most likely be severely occluded during sensing. Therefore, making hypothesis is necessary. For many methods the hypothesis are to exploit regularity of man made object by using combination of simple geometric primitives (plane, sphere, cone, cylinder ...) with strong common properties (e.g planes will tend to be parallel or orthogonal, axes of primitives will tend to be collinear), and symmetries.

(Lau et al., 2011; Umetani, Igarashi, and Mitra, 2012) add another level of constraint by stating that the object can be fabricated (e.g joins between parts must have adequate resistance and the global object must be stable). On another level (Grzesiak-Kopec and Ogorzalek, 2013; Yeh et al., 2012) use relationships between objects to define constraints that the reconstructed objects must satisfy.

Strategies for urban feature reconstruction

Because precise street feature reconstruction is quite new and connex to many research communities, we include methods with very different inputs which could be used for street feature reconstruction, even if not explicitly stated by the corresponding articles.

Some methods reconstruct directly street features (low level reconstruction), but the sensing data is sparse and often of relative low quality considering the scale of the considered objects. As it is often the case a way to simplify a problem too wide is to add constraints and knowledge about it. Some of the approach therefore add strong hypothesis about the object to reconstruct (Section 1.10.2.1).

Because reconstructing directly street feature may not be feasible, some approaches turn to classical segmentation/classification methods (Section 1.10.2.2).

This allows to decompose the reconstruction problem : First find which street object is where, possibly determining some of its properties, such as its orientation. Second, find or generate a similar 3D model and populate the reconstructed street with it.

However, finding the exact corresponding model from incomplete data for a street feature may be challenging (see introduction of this section). Therefore other methods are based on object structure analysis, decomposing it into parts. The reconstruction is then facilitated by the possibility to switch parts of the object as well as complete missing parts by a similar one (Section 1.10.2.3).

Another more radical approach, which we could call extreme classification, relies on an extensive catalogue of objects. The reconstruction process amount then to find the model in the catalogue that is the closest to the sensed object, then use the catalogue model as the reconstruction.

State of the art

Low level reconstruction

INTRO There is a great body of literature about surface reconstruction. A naive approach could be to use these methods to directly reconstruct the street objects. However due to the massive amount of occlusion (a street feature is commonly occluded halfway),

strong hypothesis about the object nature are necessary. Also, these methods do not provide semantic information about the reconstructed object (e.g a reconstructed poll wont be identified as a poll but as a cylinder).

DIRECT SURFACE RECONSTRUCTION (Bessmeltsev et al., 2012) propose a method to directly generate surfaces from 3D lines as input. The extreme data sparsity is similar to what may be available in street feature reconstruction. The authors interestingly make an hypothesis about what type of surface could be expected from a man-designed object.

Using a noisy point cloud (Guillemot, Almansa, and Boubekeur, 2012) make hypothesis on repetitions in the data to reconstruct a better surface. Their method defines local patches as small set of points. When reconstructing the surface of a patch they use the local information as well as informations of similar looking patch elsewhere in the point cloud.

SIMPLE GEOMETRIC PRIMITIVES With dense noisy point clouds of man made objects, (Li et al., 2011) assume that an object consists of regular geometric primitives globally aligned. So, they iteratively detect the primitives with the associated points that support it. Then they extract and enforce global relations among these primitives and remove the associated points from point cloud, before iterating on the reduced point cloud. ((Labatut, Pons, and Keriven, 2009; Lafarge et al., 2013)) propose other primitive-based approaches applied to buildings which may be transposed to street features reconstruction. The goal of the two works is to extract a mixture of geometric primitives and free-form mesh from noisy stereo-based point clouds. One relies on a binary space partition tree and a RANSAC detection method while the other uses a sophisticated energy-based Jump-Diffusion process.

SHAPE GRAMMARS The shape grammars like the one defined by (Krecklau and Kobbelt, 2011) generalise the simple geometric primitives. They are by construction well adapted to represent man-made objects (and even vegetation (Section1.9) Such grammars have a great generative power, but one has to resolve an inverse problem to use them for reconstruction.

This problem is solved via *the Inverse procedural Modeling methodology* (See Section 1.5.3).

Object reconstruction

RC:1.10.2.2. chap1 : urban features : object reconstruction : add ref of Timofte2011 on sewage cover detection

INTRODUCTION Given the occlusion in data, it may not be possible or satisfactory to reconstruct objects directly. Therefore many methods chose a two steps approach, where the first step detects and classifies objects in the input data. The second step can then be adapted to each object type. For each object type the options are either to reconstruct it directly using tailored methods or to populate the street with a model of this object.

Compared to low level reconstruction (Section 1.10.2.1) , these methods can be fitted to each objects, and the inserted models are cleaner than model reconstructed from scratch. A complete example of this pipeline is given by (Cornelis et al., 2008). They

use video streams from a street vehicle to reconstruct a 3D map of a city. Along the way they detect cars on the side of the road (3D bounding boxes). Ultimately, they insert into the 3D city model clean 3D car models in these bounding boxes. This greatly improves accuracy of reconstruction and realism of city model.

Classification is a transverse problem in many computer science fields. Street objects classification must be adapted to challenging input data (scale, occlusion, sparse data). Also, as stated in the introduction of this section, the number and types of street features is important. This proves to be a major obstacle for machine learning methods which rely on training data. In these training data some uncommon objects may be statistically overwhelmed by more common (and similar) objects (see (Golovinskiy, Kim, and Funkhouser, 2009)).

Another set of difficulties is added by the second step, which imposes not only to classify objects, but also to measure parameters to correctly insert models (orientation,state, potentially more parameters for parametrised objects).

We order the related methods by the detection / segmentation / classification task, the feature extraction task and the matching task. Such order is only practical because many methods mix these categories. In classification literature the word feature is often used instead of descriptor. We choose here to use the word descriptor to not introduce confusion with the topic (Urban/street feature reconstruction).

DETECTION, SEGMENTATION, CLASSIFICATION In an influential article, (Golovinskiy, Kim, and Funkhouser, 2009) use street Lidar input to demonstrate the full localisation/segmentation/classification pipeline. They test multiple classifiers methods and descriptors, and perform an experiment on large scale real world data. Their method detects around twenty different object types.

(Shao et al., 2012) also illustrate a full pipeline but not in a street object context. They use interactively segmented colors and depth images (RGBD). The extracted objects are then matched against a database of 3D models. These models are inserted using an optimisation process to determinate their size and position.

In order to tackle the scale problem, (Yu et al., 2011a) propose a segmentation of a massive city point cloud into ground and façades, and potential objects. The work of (Lippow, Kaelbling, and Lozano-Perez, 2008) adapt to the many type of objects to detect (in the computer vision field). Their method learns an AND/OR probabilistic tree for a category of object in annotated images. Such trees are then used for detection, not of one object, but of the category of this object.

Some usages do not necessitate accurate object reconstruction. For instance (Hofmann and Brenner, 2009; Soheilian et al., 2013) detect poles (respectively streets signs and markings) based on simple geometric model in order to create a landmark map which can then be used to cheaply localise other data.

Similarly, building in real time such localisation map with 3D semantic voxels (Stuckler, Biresev, and Behnke, 2012) significantly improves the registration of their colors+depth images data (RGBD). These voxel maps may also be used for more abstract task like human-robot communication.

DESCRIPTOR EXTRACTION The task of classification is often very sensitive to the choice of descriptors of an object. A good descriptor should reduce the amount of data necessary to describe the objects, but not reduce the information much. Furthermore,

the descriptors must be chosen to be differentiating between object types. A good choice of descriptors increases recognition rate and reduces errors.

We refer to the appropriate articles for the classical descriptors used by (Cornelis et al., 2008; Shao et al., 2012; Soheilian et al., 2013; Stuckler, Biresev, and Behnke, 2012) (Implicit Shape Model, simple local RGBD descriptors, many descriptors selected through Random Forest, image and contour-based).

Concerning the shape-matching methods, the choice of descriptor is of the essence. The method performance, speed, scaling and accuracy strongly rely on it. (Papadakis et al., 2007) use descriptors based on spherical projection, (Shao et al., 2011) use depth feature as well as geometric primitives, (Eitz et al., 2012) use adapted Gabor filters.

For noisy point cloud data, The work of (Kalogerakis et al., 2009) who extract lines of curvature may also be used as a descriptor for street feature. According to the authors, this curvature-based descriptor is specific to man made objects.

(Golovinskiy, Kim, and Funkhouser, 2009) outline that contextual (i.e. relational) descriptors are of great use for object classification.

In that way, (Vanegas, Bloch, and Inglada, 2013) propose a fuzzy relational descriptor that may be adapted to noisy and incomplete data. Using aerial images, the proposed method extracts fuzzy spatial relations between objects like alignment and parallelism.

A very complete generalisation of these kinds of relationships is given in (Mitra et al., 2012). This state of the art provides numerous useful reflections about the presence of total or partial symmetry in man made objects. For example, (Xu et al., 2012) propose a method to compute partial symmetries at multiple scales. Such relations could be used as high level descriptors.

MODEL MATCHING To the best of our knowledge no matching system against a 3D model database has yet been applied to street feature reconstruction. However such systems have been developed in the field of model matching. These methods may be transposed to the field of street feature reconstruction, as demonstrated by (Shao et al., 2012) for indoor objects. In fact, most of the presented shape matching methods use 2D sketch produced by a user. Nevertheless such an input could be conceptually replaced by the sensing data of street feature.

The pipeline of (Eitz et al., 2012; Papadakis et al., 2007; Shao et al., 2011; Shao et al., 2012) is similar and can be decomposed into an off-line data base creation step, and an on-line query step. First the methods extract descriptors for thousand of 3D objects and constitute a database associating object model with their descriptors. During the on-line step, an user input of a 2D drawing is analysed, the same descriptors are computed and the methods search the 3D models in the database that have the closest descriptors to the user input. The result is a list of matching shapes from database, with a matching score.

Howsoever these methods differ by the choice of descriptors, the validation ((Eitz et al., 2012) analyse the best way to perform dimension reduction (i.e. translating optimally a 3D model into 2D views)), and the reconstruction step (only performed by (Shao et al., 2012))).

(Jain et al., 2012) also perform shape matching, but in a fundamentally different way. The goal of the author is to automatically transfer materials (i.e. texture, colours and lightning) to a 3D model by matching its different parts with a 3D model database. The authors also follow the two steps that are the constitution of a database of 3D mod-

els, and then a query step. The originality is that the database is a graph of parts of models that is automatically computed based on similarities of parts (spatial, geometrical, material-wise). Querying the database then amounts to compute the graph for the queried 3D model, then add this graph to the database graph and use a loopy belief propagation algorithm to perform inference.

Interestingly such method introduces the use of structural information about objects. This information is pivotal to estimate the material of each parts.

Object structure analysis

Man made objects are constituted of parts having (potentially hierarchical) relations (symmetry, fixed angles, etc). This relations describe the object structure.

INTRO Object structure analysis may be of great help in street feature reconstruction, and this at two scales. At the part scale (decomposing an object into structured parts, e.g. a street light may be a cylinder (pole) and a sphere (light bubble)), and at the multi object scale (decomposing multi objects into structured objects e.g. a dashed marking line may be described as a repetition of aligned small pieces of plain lines.)

Such structure analysis may be useful at the object scale, because analysing the redundancy, structure and organisation of an object allows to extract higher level information about it. It can then be used to compensate noisy or incomplete data ((Shen et al., 2012) do this in a reverse way)(e.g sensing only the front part of a pole may be sufficient if we have the information that poles follow a rotational symmetry).

Moreover, a strong structural information and presence of symmetries (Mitra et al., 2012) is typical of man-made objects and may be used as descriptors for classification/-matching (Shapira et al., 2009). Alternatively, such regularities allow for compression and Levels Of Detail ((Jang et al., 2006)). This also gives an information orthogonal to pure geometric comparison: it allows to measure how similar the structure of two objects is, rather than their geometry. For instance, a motor bike and a bycicle are structurally similar, but may have very different geometries.

Secondly, some methods that leverage structure of object may be generalised at the multi-object scale, i.e. finding and using the structural relations between objects, that are known to structure the layout of objects in a street (See Section 1.7).

HOW TO DETECT SYMMETRIES Analysing the structure of a 3D object is complex because it involves unsupervised segmentation as well as a relation extraction between parts.

Among the relations used in the methods (generalised), similarities are popular.

A typical approximate symmetry pipeline is given in (Mitra, Guibas, and Pauly, 2006), where the input is a 3D model (which could also be a dense 3D pointcloud). In a first step they get random sample points from the surface, and perform pairwise symmetry parameters estimation by taking into account a patch around the points.
Then, in the space of the found pairwise-transformation, a clustering is performed to extract dominant transformations. The supports for this transformation are then computed by region growing from the sampled points.

(Xu et al., 2012) improve this process by adopting a multi-scale classification.

The authors of (Li et al., 2011) choose another direction and perform the equivalent of relation clustering with a custom graph simplifying algorithm.

Whereas partial symmetries are covered in (Mitra et al., 2012) as a generalised case of symmetries, (Vanegas, Bloch, and Inglada, 2013) incorporate them in the fuzzy logic theory. Exploiting ad-hoc fuzzy operators, they propose a way to compute fuzzy parallelism, fuzzy alignment, etc.

The work of (Cullen and O'Sullivan, 2011) generalise more the symmetry concept by constructing a tree of symmetry compositions representing a pattern. This method is very close to a procedural expression. After having computed such trees for two patterns, they can be merged to create a hybrid pattern that mixes the two input patterns.

Other methods use touching relation to extract structure.

(Shapira et al., 2009) use a custom descriptor based on local diameter of the object. They use it to iteratively fit Gaussian mixtures in order to find parts, then build a graph representing the relations between parts. They can then perform parts matching taking into account the context of the parts to match.

(Jain et al., 2012) extract structure by contact and symmetry analysis, and use it for matching or for generation of hybrid models by genetic evolution.

(Lau et al., 2011) retrieve an even more complex structure as they perform inverse procedural modelling (See Section 1.5.3). They analyse contacts between parts of an input 3D model, parsing it into a graph of connections. Then they use a custom grammar to express this graph by inverse procedural modelling. Using the grammar with the extracted rules and parameters generate a fabricatable 3D model.

For completeness sake we mention that some methods consider the decomposition of object into parts as a preprocess step that has already been performed ((Chaudhuri et al., 2011; Shen et al., 2012; Xu et al., 2012)).

USING THE STRUCTURE At the object scale having such a structural description of objects allow (Shen et al., 2012) to match parts of 3D model on noisy and sparse RGBD point cloud. The authors of (Chaudhuri et al., 2011) tackle another problem by suggesting parts when building a 3D model from scratch. Yet their method may be used to complete occlusions resulting from street feature sensing.

Expressing the object structure is not necessary if the goal is to respect symmetry relations between parts of 3D models when editing (structure preserving editing). For instance (Bokeloh et al., 2011; Bokeloh et al., 2012; Gal et al., 2009) analyse a 3D model to detect symmetries (respectively more general patterns), which produce a set of constraints that are linearised, allowing to edit the shape interactively while computing a solution respecting the constraints by propagating the changes and minimising an energy locally (respectively minimising an energy).

At the multiple objects scale, (Krecklau and Kobbelt, 2011) propose an extension to their grammar that add the possibility to model interconnected structures, which are common in street. However using such grammar would require to extract relations and patterns amongst street objects. In the same field (Grzesiak-Kopec and Ogorzalek, 2013) adapt a shape grammar to resolve a layout problem.

Other methods uses these high level data that model the relationship between objects. Still, in all the article we present these relational data are user input and are not extracted (with the exception of (Fisher et al., 2012)).

Putting in leverage these relations allows to use powerful optimisation methods to generate a good placement for furniture in a room in ((Yu et al., 2011b)). One limitation is that the number of objects is fixed.

(Yeh et al., 2012) remove this restriction by proposing a similar method that uses another advanced optimisation framework to find conjointly the number of objects, as well as their position and orientation.

Those two methods could be used in street object reconstruction by resolving an inverse problem : given noisy observations and relations, find an optimal objects reconstruction.

Interestingly, (Fisher et al., 2012) directly extract relationships between objects from a clean 3D indoor scene using Bayesian networks and Gaussian mixtures. In a further step they generate a new scene with objects matched from database satisfying the extracted relationships.

The relationship between street features is discussed in detail in the section 1.7.

Conclusion

Urban features are important (number, role). Urban features are strongly dependent on context (a same white marking could have totally different meanings if it were on the road surface or on the sidewalk). Reconstruction is difficult because data is sparse, yet because the objects are man-made, many hypothesis can be made. When this is not sufficient, user interaction is necessary. Many reconstruction strategies are possible, from direct reconstruction, to model oriented reconstruction, to procedural modelling and grammar, to use of catalogues of objects.

CONCLUSION

In this state of the art we tried to consider all aspects of urban modelling/reconstruction (street, street network, vegetation, urban feature). Each one of this aspect has a dedicated conclusion (Sec. 1.7.6 on page 31,Sec. 1.8.3 on page 35,Sec. 1.9.4 on page 40, Sec. 1.10.3)

There are common elements for all these aspects of urban reconstruction. The first element is that each aspect is important for urban reconstruction. We can not simply reconstruct buildings to reconstruct a city, other aspects also have to be reconstructed.

The second element is that reconstruction is difficult for each aspect, the challenges come from the complex nature of urban environnement and from the limitations of available data.

The third element is that all the aspects of urban reconstruction seem to be linked. Street network reconstruction require information about urban feature, which are influenced by street morphology, which influence urban vegetation.

The last element is that many strategies are available to reconstruct each aspect, from direct reconstruction to procedural modelling. (Inverse) Procedural modelling seems to have potential to reconstruct all the aspects.

2

2

PCS : A POINT CLOUD SERVER TO MANAGE POINT CLOUDS

Performing reconstruction implies to use observations of the objects to be reconstructed. In our case, we have access to advanced street level data thanks to a mobile mapping system (Paparoditis, Craciun, and Schmitt, 2012). Those data include many street view images and street Lidar files. Lidar data are especially difficult to deal with, because they can be very large (Billions of points), with multiple complex overlaps (the same object may be sensed at different distances at different times depending on the vehicle trajectory, occlusion, etc.), and because they contain low level information (points with few physical attributes).

We chose to integrate those points into a database server. The advantages are to have all the relevant data (vector, raster, Lidar points) in the same place as the road modelling, which allows to combine those data to exploit the Lidar point cloud. We also created in base processing algorithm of Lidar Point cloud, such as urban feature detection, which are therefore executed directly within the database (road marks detection, kerb detection, road network reconstruction).

We propose in-base Levels Of Detail (LOD) construction and usage to parametrically reduce the number of points. We also extract a dimensionality descriptor from our LOD method, and demonstrate its interest by using it to perform classification. Due to size considerations, those subjects are developed in an independent setting (Appendix 7).

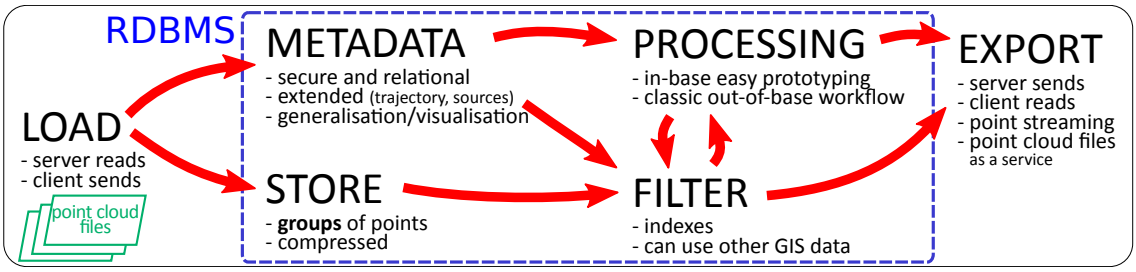


Figure 19: Graphical Abstract : In-base point cloud management pipeline in the Point Cloud Server (PCS).

ABSTRACT

In addition to the traditional Geographic Information System (GIS) data such as images (rasters) and vectors, point cloud data has become more available. It is appreciated for its precision and true three-Dimensional (3D) nature. However, managing point clouds can be difficult due to scaling problems and specificities of this data type. Several methods exist but are usually fairly specialised and solve only one aspect of the management problem. In this work, we propose a complete and efficient point cloud management system based on a database server that works on groups of points rather than individual points. This system is specifically designed to solve all the needs of point cloud users: fast loading, compressed storage, powerful filtering, easy data access and exporting, and integrated processing. Moreover, the system fully integrates metadata (like sensor position) and can conjointly use point clouds with other geospatial data, such as images, vectors, topology and other point clouds. The system also offers in-base processing for easy prototyping, parallel processing and scales well. Lastly, the system is built on open source technologies; therefore it can be easily extended and customised. We test the system will several *billion* points from point clouds from Lidar (aerial and terrestrial) and stereo-vision. We demonstrate *~400 million pts*/h loading speed, transparent-for-user and greater than *2 to 4:1* compression ratio, filtering in the approximately *50* ms range, and output of about a million pts/s, along with classical processing, such as object detection.

The last decades have seen the rise of GIS data availability, in particular through the open data movement. Along with the traditional image (raster) and vector data, point clouds have recently gained increased availability (the site opentopography[1] is a good example) and usages (robotic, 3D, virtual reality). Sensors are increasingly cheap, precise, available, and the point cloud complements images naturally. Point clouds are un-organized, high geometric, low feature precision data, where images are organized low geometric high feature precision data, which makes point clouds dual-like to images. However, due to their massive un-organized nature (no neighbourhood information) and limited integration with other GIS data, the management of point clouds still remains challenging. This makes point cloud data barely accessible to non-expert users. Yet many fields would benefit from point clouds had they an easiest way to use them.

Problems

Point clouds data sets are commonly in the TeraByte (TByte) range and have very different usages; therefore every aspect of their management is complex and has to scale.

Having such large data sets makes the compression an essential need. Not only is the compression necessary, but it also has to maintain a fast read and write access, and be transparent for the users. Indeed, we observe that today virtually all images and videos are compressed; most users not noticing it at all.

Similarly, so much data cannot be duplicated and must be shared, following again a broader trend in the Information Technology world. Sharing data necessarily introduces concurrency issues (several users simultaneously reading/writing the same data).

Users usually need to access only a part of the data at once, thus efficiently extracting (filtering) a subset is important. With many varying usages, the criterae for choosing the subset may be volatile, and sometimes mixed.

Visualising something helps understanding it. In the case of multi-billion point clouds, a Level Of Detail (LOD) strategy is necessary, because the data set cannot be displayed in its entirety at once.

Features of point clouds can be very different depending on their source (Lidar, stereovision, medical ...), regarding the number and type of attribute, the geometric precision and noise, etc. Yet, point clouds usually are geospatial data, which makes them akin to vectors and rasters from the Geographical Information System (GIS) world. Thus, point clouds may be used conjointly to other data types, either directly or by converting point clouds to images or vectors.

Lastly, point clouds are processed in many different ways suiting each user need. These methods must be fast and easy to design, scale well, and be robust.

Another important problem is related to point cloud management. For various reasons point clouds are often handled as sets of points. Yet, a point cloud (data set) is much more than points, as it also includes metadata and other informations like sensor geometry, etc. Managing those data sets is difficult; like knowing which data sets are available and where. Because data sets are heterogeneous, managing extended metadata such as point cloud coverage, date of acquisition and so is also difficult, especially

1 www.opentopography.org

without a standard data format. Treating point clouds as only points is especially problematic, as is illustrated by a very recent benchmark release[2], which provides massive and very useful hand-labelled point clouds, yet does not provide any meta data at all, neither extended meta data nor contextual data.

In this article, we propose to use a point cloud server (PCS) to solve some of these problems. The proposed server architecture provides perspective for metadata, scalability, concurrency, standard interface, co-use with other GIS data, and fast design of processing methods. We create an abstraction layer over points by dealing with groups of points rather than individual points. This results in easy compression, filtering, LOD, coverage, and efficient processing and conversion.

Related work

FILE SYSTEM Historically, point clouds have been stored in files. To manage large volumes of these files, a common solution is to build a hierarchy of files (a tree structure, like a quad tree) and access the data through a dedicated set of softwares. This approach is continuously improved (Hug, Krzystek, and Fuchs, 2004; Otepka, Mandlburger, and Karel, 2012; Richter and Döllner, 2014) and a detailed survey of the features of such systems can be found in (Otepka et al., 2013). This approach is simple, and scaling is relatively easy (provided the Operating System (OS) maximum file number is not reached). However, using a file-based system has severe limitations. These systems are usually built around one file format, and are not necessarily compatible. Recent efforts have been made towards format conversion[3]. The features of such systems are very limited (limited meta-data handling, lack of integration with other GIS data, difficulty to use several point clouds together). Moreover, these systems are not adapted to share data and multi-users usages (concurrency).

DBMS FOR POINTS Hofle, 2007 proposed to use a Data Base Management System (DBMS) to cope with concurrency. The DBMS creates a layer of abstraction over the file-system, with a dedicated data retrieval language (SQL), native concurrency capabilities (supporting several users reading/writing data at the same time), and the wrapping of user interactions into transactions that can be cancelled in case of errors. The DBMSs have also been used with raster and vector data for a long time, and the possibility to define relations in the RDBMSs (Relational DBMS) offers a simple way to create robust data models, and deal with meta-data. Adding the capacity to create point clouds as services, DBMSs solve almost all the problems we face when dealing with point clouds. Usually, the database stores a great number of tables, and each table stores a point per row (Lewis, Mc Elhinney, and McCarthy, 2012; Rieg et al., 2014). Such a database can easily reach billions of rows. Nevertheless, storing these many rows is problematic because DBMSs may have a non-negligible overhead per row. Moreover, indexing those number of row is slow and takes a lot of places, and the possibility of compression are limited.

COLUMN STORE DATABASE AND NO-SQL These limitations are more generic than for point cloud usage, and apply to any massive amount of data which is weakly rela-

2 www.semantic3d.net
3 http://www.pdal.io/

tional and does not get updated often. As such, they have been researched and inspired the concept of column-oriented databases, such as MonetDB[4]. This database is used to store individual points (Martinez-Rubi et al., 2014, 2015; van Oosterom et al., 2015). Thise approach is effective to store large amounts of row without much overhead, and also solve most of the indexing issues. However points are not compressed, integration with other GIS data is weak, and scaling to multiple computers is not straightforward . In parallel, stripped down column stores were proposed, having been specially tailored for massive and weakly relational data,forming the No-SQL databases. They scale extremely well to many computers and can deal with large amount of data (Wang, Aji, and Vo, 2014 and SpatialHadoop[5]). However, this comes at a price. NoSQL databases must drop some guarantees on data. They are not integrated with other GIS data and have much less functionalities. Indeed, NoSQL databases are closer to being a file-system distributed on many computers (with efficient indexing) than being DBMSs. Thus massive scaling still necessitates specialised hardware, and the people to maintain it.

CLOUD COMPUTING A recent possible workaround for this issue is to use Cloud Computing facilities [6] to store the points, like Amazon S3. In this solution, data storage is offered as a service and externalized. This may provide the ultimate scaling, but it suffers from the same aggravated limitations as the NoSQL, with open issues on indexing.

DBMS FOR PATCH All the previous data management systems try to solve a very difficult problem, managing a massive quantity of individual points. The solutions that scale well must focus on data storage and retrieval, and drop the rest of the management problem (feature, metadata, integration, processing). Another recent approach is being explored in pgPointCloud, 2014 and other commercial RDBMS. The key idea is to manage groups of points (called patches) rather than points in a RDBMS. Creating this abstraction layer over points allows retention of all the advantages of an RDBMS, but keeps the number of rows low, thus avoiding the associated scaling difficulties (index, compression). Moreover, the proposed abstraction offers new theoretical possibilities, because it creates generalisation of the groups of points. The price is that to access a point its whole group has to be accessed first, and so the way points are grouped must be compatible with intended point usages.

In this work, we present a point cloud management system fully based on pgPointCloud, 2014 and open source tools. We test this system in every aspects of point cloud management to prove that it answers all the global needs of point cloud users, as illustrated by Figure 19.

Plan

Following the IMRAD format (Wu, 2011), the remainder of this article is divided into three sections (Method: Sec. 2.3 on page 57, Result: Sec. 2.4 on page 68, Discussion: Sec. 2.5 on page 83)

4 www.monetdb.org
5 http://spatialhadoop.cs.umn.edu/
6 https://github.com/hobu/greyhound

Each section has the same organisation covering the bases of point cloud usages (See Figure 19 on page 52). First we consider how points can be stored as groups in a Point Cloud Server (Storing). Then we consider how to load point clouds in the PCS (Loading). Point clouds contain metadata that can also be stored and used (Point Cloud Context). We study how to access only a part of the points using conditions (Filtering). Points can be outputted from the PCS (Exporting). Last we consider various methods to exploit points (Processing).

Thus, each subsection is found in method, results, discussion. For instance, "Loading" is in *method* : Sec. 2.3.2 on page 59 , *result* : Sec. 2.4.2 on page 73, *discussion*: Sec. 2.5.2 on page 83.

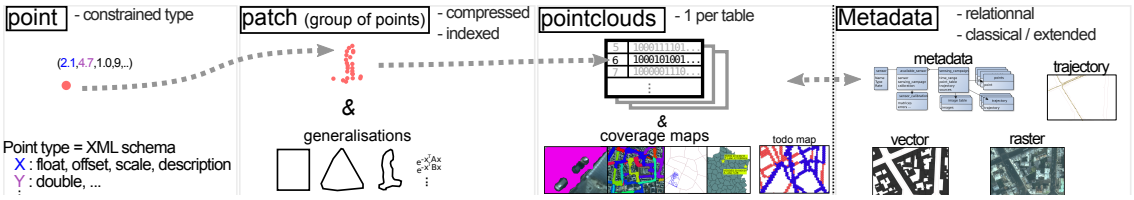


Figure 20: PgPointCLoud storage illustration. Point attributes are described by an XML schema. Points are grouped in patches, indexed and compressed, which may have several generalisations. A point cloud is stored in a table, with one patch per row, along with other table generalising the point cloud (like coverage map). The PCS also store metadata (date, place) in a relational way, extended metadata like trajectories, and possibly other GIS data like vector and rasters.

METHODS

Storing groups of points in a RDBMS

The proposed solution relies on a team PostgreSQL, 2014 RDBMS server using the team PostGIS, 2014 and pgPointCloud, 2014 extensions. The key idea is to store a point cloud per server table, with one row storing a compressed group of points. Groups of points are called patches of points. The type of a point (attributes size, definition, nature) is defined in a global XML schema. See Figure 20 for an overview of how storage is organized in PCS.

The user can load data into the server by several common means (using major programming languages, Bash, SQL, Python), from any format of point cloud that can be expressed as a list of values. Point clouds are stored without loss and are compressed. The very sophisticated database indexes allow efficient filtering of the patches. Point clouds can be used with vector and rasters and other point clouds. Metadata are integrated and exploited. Furthermore, point clouds can be easily converted into other GIS data (vector/raster). Processing methods are directly accessible within the database; additional methods can be added externally or internally. Accessing points from the database is also easy and can be done in several ways (whole files, specific points and streaming).

Storing groups of points rather than points

Briefly, storing groups of points offers the advantages of generalisation (potentially more complex semantic objects), reduces the number of rows in the database by several orders of magnitude, reduces index size, allows efficient compression, and offers a common framework for different types of point clouds coming from different sources. Working on groups of points separates the filtering and retrieving of points. This allows to take decisions based on filtering results before retrieving points. For instance based on the density, an optimal LOD can be automatically chosen. Groups can also be easily split or fused at any point after data loading.

It is important to note that storing groups of points rather than points also introduces a fundamental limitation : to obtain an individual point, we need to get the full group

first. This means that the grouping points approach is only possible when points can be categorised into groups that are coherent for the intended applications. Incidentally, all intended applications must require the same grouping rules.

GENERALISATION Choosing to use groups of points instead of individual points amounts to use a generalisation of the data, that is an abstraction. Abstracting the data is very common in GIS. For instance, when making a map of a big city, representing all individual building footprints would diminish the user understanding of the map. Instead, building footprints may be aggregated to form urban block. (see Mackaness, Burghardt, and Duchêne, 2014 for a recent introduction to the generalisation topic).

Regarding point cloud, we may have a group of 10k points sampled along a small part of the road that is flat (10k 3D points). For some application, we could abstract the group with a plan (three 3D points). Geometrically representing this group of points by a plan reduces storage, but the change is more profound, because the plan is another representation of the underlying object that has been sensed.

The plan could be used as part of a facade reconstruction like Lafarge et al., 2013, or even be the base for a further building generalisation as in Meng and Forberg, 2007.

The generalisation does not have to be geometric. For instance a group of points can be abstracted by statistical distributions (similarly to Preiner et al., 2014, although they use the distribution for surface reconstruction.)

Ummenhofer and Brox, 2015 illustrate both this usage. They use an octree and thetrahedrals as support for their geometric generalisation, and aggregates as a statistical representation. Combining both, they can reconstruct surface without using the points but only the generalisations.

Such generalisation is by essence higly tailored to an usage, being a form of information-loosing modeling. In this article we propose several generalisations adapted to a variety of usages (Figure 37 page 76). Those generalisations can be used conjointly in the Point Cloud Server. By doing so, we avoid the pitfall of duplicating the data for each specific usage.

Point grouping strategy

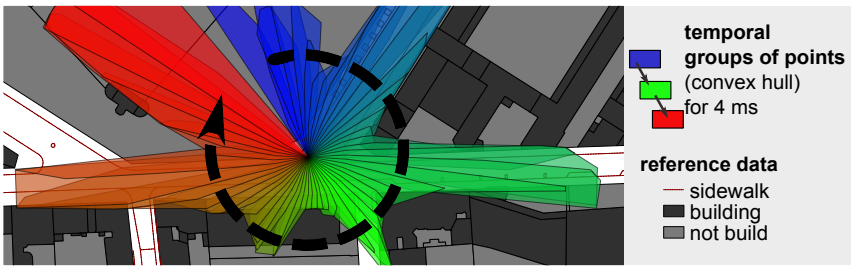


Figure 21: Rotating Lidar (Velodyn) with strong temporal dependency (200 ms acquisition).

Points should be grouped in regard to how they will be retrieved afterwards. As points tend to be retrieved by their spatial position (spatial-grouping), grouping the points that are spatially close together makes sense. Some Lidar devices include a strong time dependency, and are commonly used to detect moving objects. In this case time-grouping may be interesting, in order to differentiate easily between points roughly at

the same position, but acquired at varying time (See Fig. 21, and Section 2.4.1 and Fig. 32 page 70). Grouping rules may also mix spatial and temporal rules, as well as other rules like semantic grouping if this information is available (for instance, points pertaining to buildings would be in separate groups than points pertaining to the ground).

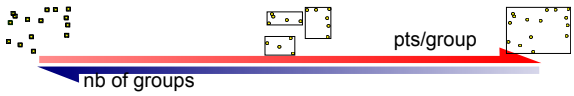


Figure 22: Choosing group size is a trade-of between filtering and storage.

While many rules are possible to group points, it always results in a trade-off (see Figure 22). Small groups means a lot of groups, which is bad for storing, because it will increase the number of rows, thus the size of the indexes and associated overhead, and reduce compression possibilities. On the opposite having big groups is bad for filtering (and maybe compression, if points becomes too dissimilar), because to get one particular point, the whole group has to be read.
However, we stress that all groups do not have to follow the same rules, thus group size can be adapted to local characteristics of the point cloud, for instance to geometry (grouping depending on density) or semantic (grouping depending on attributes or classification). See Section 2.4.1 for an example of varying grouping size based on geometry, where groups are merged/split in 8 (similarly to voxels) until the target number of points per group interval is met.

Loading

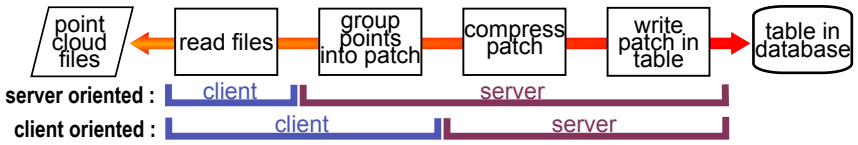


Figure 23: Processes necesary for load (and output) of points into the database can be performed by the client(s) or the server, depending on the application.

Writing data in a PostgreSQL RDBMS is standard. Clients exist in all major programming languages. Because DBMSs are built for concurrency, all presented methods use parallelism.

For the specific application of writing point cloud , the goal is to go from point cloud files to (compressed) patch of points stored in tables inside the database (See Figure 23). To this end, several intermediary steps have to be performed. In a server/client architecture, we conceptually separate solutions depending on who is supposed to cover most of the process. In a "Server oriented" design, the server does almost everything. In a "Client oriented" design, the client does almost everything. Please note that this division is only conceptual.
Section 2.4.1 page 70 contains more details about how patches are compressed.

PARALLEL LOADING ('SERVER ORIENTED') Our first loading method (Figure 24) reads point cloud files, convert them to a stream of attributes and writes them to tem-

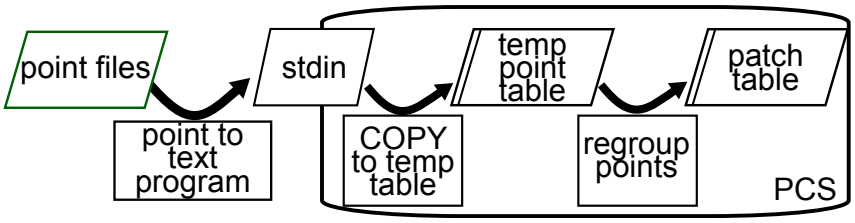


Figure 24: Conceptual schema for parallel loading.

porary tables in the database. The database groups points into patches and adds the patches to the final point cloud table. Please note that the database could directly read point cloud files.

DISTRIBUTED PARALLEL LOADING ('CLIENT ORIENTED') In previous method, the database performs the grouping of points into patches and the actual writing of patches into tables. We could lessen the workload of the server by allowing the client to do the grouping.
We design a loading method of type 'client oriented' (Python).

It is similar to the method adopted by the PDAL[7] project, which we also test. The clients read point cloud files, group points into patches, and send the patches to the database. The database compresses the patches and writes them into the final point cloud table. Please note that the client could also perform the compression.

Point Cloud and Context

Historically, RDBMS databases have been designed to create and maintain relationships between data. Because our method relies on a RDBMS, we can leverage this capacity. One of the goals of our system is to manage point clouds rather than points. By that, we intend that a point cloud is not a set of points, but rather a set of points associated with various meta-data and contextual information (and maybe processing methods). In particular, our system can store the full meta-data model, as well as more indirect meta-data like the trajectory of the sensor. Meta-data can also be organised in a relational way to be coherent between different point clouds. By integrating point cloud into a relational database, and having several representation for patches (See generalisation concept, in Section 2.3.1.1), we can easily create coverage maps. It also enables to use several point clouds together as well as mix point clouds and other GIS data (raster and vector), directly or after converting point clouds to other GIS data types.

MANAGING METADATA The point cloud server offers the perfect framework to regroup all the metadata concerning the point cloud. For instance, the popular .las file format proposes to store a project id, date of acquisition, and name of the hardware. Being based on standardized and limited fields, very few metadata are stored. Furthermore, the information can be missing or erroneous. Using a RDBMS, it is possible to create an unlimited relational model of the metadata and to easily enforce it (automatically). For instance, instead of a project id, we could refer to a list of projects, each

7 www.pdal.io

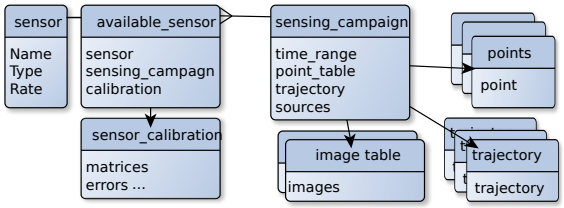


Figure 25: Example of a data model to store metadata. See Hofle, (2007, p. 15) for a real model.

having an associated list of partners, start and finish dates, associated authorizations, etc. Instead of the name of the hardware, we could refer to a list of hardware, having each different configurations, typical precision of the sensor, point type, methods for reading the raw data, etc. See Figure 25 for a basic example of metadata schema. We stress that in fact several point clouds metadata are already related to each other in an implicit way. For instance the date of acquisition of a point cloud can be implicitely related to the date of acquisition of another point cloud.

The benefits for point cloud management are numerous, from simple such as looking for point cloud based on those metadata (e.g. find all point clouds in a given area with given density, acquired in given time range, whose geometric error is less than 1 cm), to much more complex, such as on-the-fly re-registering of the point clouds when the estimated sensor position is updated.

Such metadata could also be used in the filtering step. For instance, for an application relying on colour, the user would be able to automatically get points acquired through stereo-vision and not through Lidar.

In a more generic way, we point to the success of the Resource Description Framework (RDF) in the last decade as a sign that metadata management is important and expected by users.

COVERAGE MAP Using the server, we create metadata-like point clouds coverage maps (Figure 35, 36, page 74) , which are essential to quickly understand point clouds coverage , similarly to Lewis, Mc Elhinney, and McCarthy, 2012, Figure 8, or Youn et al., 2014. The idea is to have several representations of a point cloud. The 2-level representation would be to represent the area covered by the point cloud by polygons at a detailed level (large scale). For performance and map-generalisation reasons, the point cloud could also be represented by a single point when viewed from afar (love scale)

With this set of maps, one can instantaneously and easily check what type of point cloud is available in a given area using a colour code (for instance). Because the Point Cloud Server mixes GIS and point clouds, going a step further than the two-levels visualisation (a point at small scale, detailed polygons at large scale) toward a 4-level visualisation is natural. More generally, mixing Remote Sensing data and GIS data enable much more advanced applications (Aubrecht et al., 2009). See Section 2.4.3.1 for full details on coverage map creation.

EXTENDED METADATA We can extend the classical notion of metadata a step further and consider that it also concerns the raw information that was used to create the point cloud. For Lidar point clouds, this would be the trajectory and position of the sensing device, along with the raw sensing files. For stereo point clouds, this would

be the camera poses for every image used to construct the point cloud, along with the images. This information can be stored in the server, and leveraged in filtering (see Section 2.4.3.3), processing and uncertainty management (for instance registration).

USING SEVERAL POINT CLOUDS AND OTHER GIS DATA Point clouds are created by different sources, like stereo-vision, aerial Lidar, terrestrial Lidar, RGBZ device (Kinect), medical imaging devices (MRI), etc. The Point Cloud Server mixes all this data, along with other GIS data (rasters and vectors). Vectors and rasters are stored and exploited using team PostGIS, 2014. We can use geo-referenced point clouds together, thus exploiting their complementarity.

In a typical scenario, a user interested in a place queries the Point Cloud Server. He automatically obtains points from the several point clouds available at this place, for instance a low resolution, large coverage 1 pt/m² aerial Lidar point cloud, complemented by a more detailed but very local 10 kpts/m² stereo-vision point cloud.

POINT CLOUD AS RASTER OR VECTOR In the spirit of generalisation (see Section 2.3.1.1), it is advantageous for some applications to convert points to other GIS data types, such as raster or vectors, directly within the database. We propose several in-base groups of points vector representations, such as bounding box, oriented bounding box, concave envelope similar to alpha shape (Edelsbrunner, Kirkpatrick, and Seidel, 1983), and bounded 3D plans (Figure 37). These representations can be used to extract information at the patch level, accelerate filtering, enable large scale visualisation, etc. We also propose two in-base means to convert points to multi-band rasters by a Z projection.

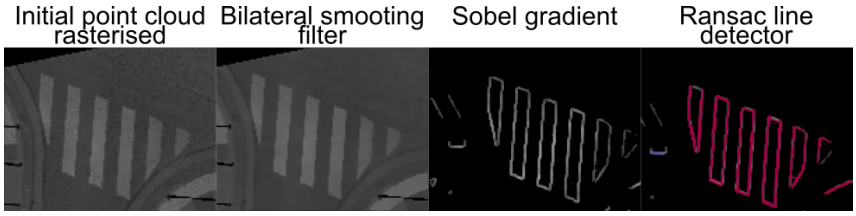


Figure 26: A part of a point cloud is converted to a raster. We use bilateral smoothing, gradient (Sobel), and line detection (RRANSAC by Chum and Matas, 2002) to reconstruct the pedestrian crossing. These operations are much faster and easier on rasters rather than points.

Rasterisation is a common first step in the literature because neighbourhood relationships are explicit between pixels, unlike points. Figure 26 illustrates how a first conversion to raster allows to use powerful and classical image processing methods to extract information from point cloud.

POINT CLOUD PATCHES AS GRAPH / TOPOLOGY The specificity of point cloud is to not embed neighbourhood information. Yet graph analysis offers powerful tools. We propose to take advantage of team PGRouting, 2015, which is a PostgreSQL extension that adds basic graph functions.

We can build a weighted graph embedded in 3D over the groups of points (ie. a graph whose vertices are the groups of points and edges the neighbouring relations

between those groups, while the weight of an edge is the 3D distance between centroids of groups of points). (See figure 27)

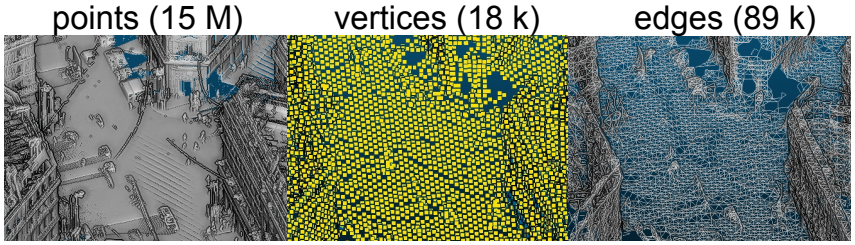


Figure 27: Building a graph embedded in 3D over point groups. Up-Left: the original part of the point cloud. Up-Right: the node of graph (centroid of patches). Down-Left: graph edges: the adjacency relationship between patches. Down-Right: visualisation in GIS.

This graph is in fact a very rough approximation (up to the patch size) of the point cloud surface. We can leverage it for fast geodesic distance computation along this surface.

Orthogonally, we can build a simplified graph and use it as the starting point of road network reconstruction. Road network reconstruction is a large topic with widely different types of strategy (see Quackenbush, Im, and Zuo, 2013 for an introduction), we only use this topic to show the PCS capabilities, (see Section 2.4.3.6 and Figure 42 page 77).

Point Cloud Filtering

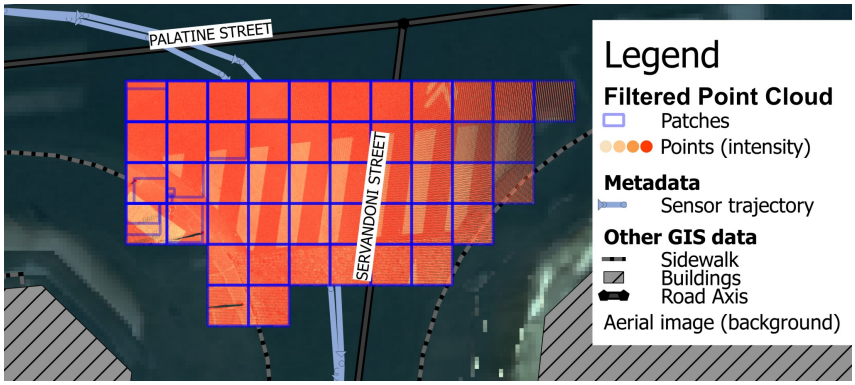


Figure 28: Illustration of a filtering conditions presented in Section 2.4.3.3 p.75. Among billions of points, only those respecting complex filtering conditions are found in ~ 0.1s. Results are shown in QGIS.

Point clouds are big; yet, we often need a very small part of them (Figure 28, parameters in 2.4.3.3 page 75). Thus, the capacity to filter a point cloud is essential for many uses. Acceleration structures (commonly called index) are the accepted solutions. This essentially creates indexes of the data to accelerate searches. Octree, B-tree, R-tree, and Morton-curves are popular acceleration structures. Designing and optimising these indexes is a major research subject (see (Kiruthika and Khaddaj, 2014), for instance) and is also the main designing factor in point cloud management systems.

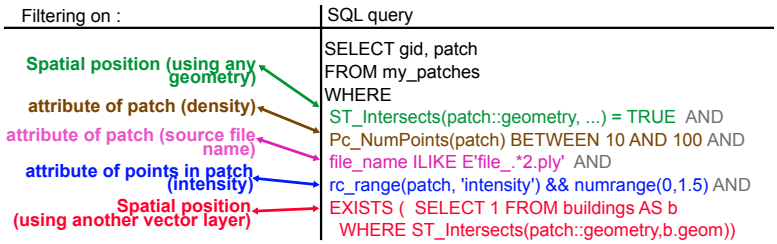


Figure 29: Example of a filtering query on patches with various type of filtering conditions.

FILTERING STRATEGY Because our system stores patches (groups of points), we can separate the filtering and the retrieving of data. The strategy is then to first efficiently filter data at the patch level by rejecting most of the patches (reducing points from billions to millions, for instance), then, if necessary, further filter the remaining points.

INDEXING Our system extensively uses n-D indexes (BTree, RTree) that are native to PostgreSQL. We index patches (not points). Basically, these indexes answer in about 0.01s to any filtering queries, such as 'What are the patches with $f(patch)$ between .. and ..'; provided that $f(patch)$ is appropriatly indexed. $f(patch)$ can be anything, a spatial position, an attribute of the points contained in the patch, a function, etc.

Indexes of functions are very powerful and can save a lot of space (no need to add an extra column to store the value of f for every row). For instance, we may have a fast function f that gives a measure in $[0;1]$ of how much the patch looks like a vertical cylinder. Now, when looking for all the patches p that really resemble cylinder ($f(p) > 0.8$), for instance), we do not need to recompute f each time for every patch, nor store all values of f. The server will only store simplified f values (typically stored on fewer bits) within the index, use it to remove the vast majority of useless patches, then recompute f for the few remaining patches that are good candidates.

PostgreSQL also determines whether to use available indexes or not, and in which order. This feature may prove essential; indeed when accessing a large amount of information, it will be faster to simply read all the data rather than use the index (sequential vs. random access). This decision is automatically made based on statistics on tables and a genetic optimisation engine. To give an idea, the database can estimate how much rows will be needed by a query. Then, knowing the cost of reading one row via index (random access), and the cost of reading the whole table (sequential access), it can decide which strategy to choose. The genetic optimisation engine is used to choose how the query will be executed, using join, index, inner loops, etc.

Figure 29 illustrates a filtering query combining various conditions.

Exporting

Similarly to Section 2.3.2 (Input, page 59), we divide the output methods into two categories. The first family of solutions is when the server perform most of the output processes ("Server oriented"), for instance writing the points in a file, or letting the user access the points through queries.

The second family of solution is when the clients perform most of the output process ("Client oriented"). We can also see the output as a service, be it for classical GIS software (using the lens) , or for WebGL browser.

PLY FILE AS A SERVICE (PLYFAS) 'SERVER ORIENTED' Our system can be used transparently with a file-based workflow. Indeed, users may already have legacy processing tools that work with files. Of course, these tools could be easily adapted to read points from the database and not from files, but users may want to use their usual tools as-is. For this case, we propose PLYFAS, an easy means to export points from the database and create a .ply file (please note that PDAL could also be used to export PLY files). The user can use the small PLYFAS API to request the database to create a ply file from any set of points. The user may simply want one of the exact original point cloud files that were loaded into the point cloud server (mimicking a traditional workflow). However, the user has also access to much more power and can request a file with filtered points by any means introduced in Section 2.3.4, or with the additional processing results of Section 2.3.6. For instance, the user can request all the points in a given area that have been classified as 'building parts' with a given confidence, and that were sensed during the second week of March 2014. In addition, the user can ask to get a target point density (Level Of Details), and to get deduplicated points (duplicated points are removed from the result), etc. Some of this operations are easy to perform in SQL (See for instance Section 2.4.3.6 page 79).

USING POSTGRESQL DRIVERS/CONNECTOR ('SERVER ORIENTED') PostgreSQL can be accessed using many programming languages, thus any PostgreSQL driver can be used to connect to the server and output points. This work-flow is very similar to what a classical processing program would do. The classical *'open point cloud file, read points, do processing, write results'* becomes *'connect to server, read points, do processing, write results on the server or elsewhere'*.

By using the server to access points, the user gets additional capabilities. For instance the user does not have to read a whole file (or any files) if he is interested in only a few points. Using the point cloud server, the user can directly filter the point cloud to obtain only the points desired, and even use in-base processing or LOD to further limit the points obtained. Furthermore, the PCS can be accessed concurrently by several users, facilitating parallel processing, and more notably parallel writing (for the results for instance) in the server.

MASSIVE PARALLEL EXPORT ('CLIENT ORIENTED') We also designed a Python method to perform massive parallel export. Similarly to Section 2.3.2 (Input), the goal is to reduce the work done on the server and increase the work done on the clients. In this version, the server sends raw binary uncompressed patches (groups of points), and the transformation to points is done by the client(s). This is similar to PDAL workflow.

LENS FOR TRADITIONAL GIS Point clouds are best visualised in dedicated software. Yet, point clouds are also geospatial data, and benefits much from being visualised and analysed in powerful GIS tools (like QGIS). However those tools do not scale over the Million points range. We propose a "lens" that reveals the points it covers (Figure 30)). The idea is that a user moves a polygon acting as a "lens" over a place of interest in the

map, revealing the underlying points, using any GIS client that can access the database. The concept has already been used to improve an interface (See Bier et al., 1993; Lobo, Pietriga, and Appert, 2015), Pindat et al., 2012 also proposed a lens with varying shape. Using triggers and a view, we ensure that the points are updated when the lens changes. Moreover, the lens also allows to choose the density of points it display (using Level Of Details), and the vehicle pass we are interested in (temporal filtering). This feature is necessary, because the registration error between several passes may be a problem (See Figure 30).

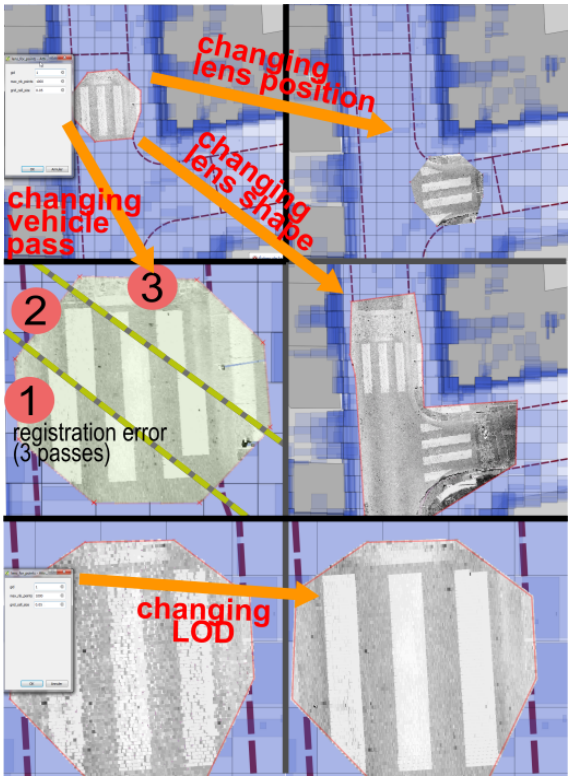


Figure 30: A polygon representing a lens that reveals points underneath it (among billions), with a given Level Of Detail and vehicle pass. Points are automatically updated when the lens if modified by the user, using a pure in-base solution, which makes it compatible with any client.

ASYNCHRONOUS POINT CLOUD STREAMING TO BROWSER ('SERVER ORIENTED') The last output possibility is to stream points in a web context. The goal is to display a point cloud into a web browser with background loading (*i.e.*, the points are displayed as they arrive, the user keeps browsing and the loading is non-blocking).

For this, we use a Node.js server between the client and the Point Cloud Server, which enables non-blocking interactions. We stress that from the PCS perspective, the task is standard (give points that are at a given place), again possibly taking avantage of LOD (send only the necessary number of points, and not all the possible points, which may be critical for limited-hardware situation like mobile phone or tablets). We use an implicit LOD scheme which is described in Cura, Perret, and Paparoditis, 2016 (Working paper).

PROCESSING POINT CLOUDS We think it is important to offer both point clouds and adapted tools to users (leaning toward giving access to services). Indeed, for most of the users, a point cloud is only a mean to get another information via processing.

Our system can be used for processing in two ways. The most classical is out-of-base processing. A client obtains the points, does something, and writes the results in the server or elsewhere.
However, our system also offers in-base processing. In this form the user directly adds processing methods within the PCS. Processing methods become very close to the data and can be reused or combined to create more complex methods. The client does not have to install anything (all methods live within the server), which is more practical (version management, dependencies ...).

An advantage of in-base processing with the PCS is that it is easy to add new processing methods. These methods can be written for efficiency (C, Cpp) or using high level languages (Python, R) for very fast prototyping. We determined that the most useful in-base processing functions should be fast and simple. This way, the newly written functions can be used in other aspects of the point cloud server, such as indexing, or be combined directly in SQL queries. For instance,

```
SELECT classify(extract_plan(patch), extract_feature_1(patch),...)
FROM patches
WHERE compute_verticality(patch)>0.8
```

Of course both type of processing can be used conjointly. In the previous scenario, the classifier would be trained outside of PCS for better memory and performance management.

General System Test

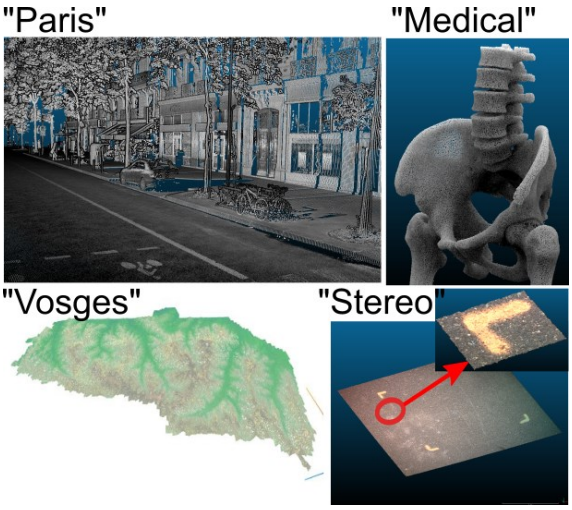


Figure 31: Paris data set (terrestrial Lidar), Medical Imaging data set (X-Ray CT Scan), Vosges data set (aerial Lidar), Stereo data set (Stereovision)

We design several experiments to test all components of the point cloud server on several datasets (Figure 31). All experiments have ample room for optimisation, and can be easily reproduced (open source tools). We use PostgreSQL 9.3, PostGIS 2.2, Pg-PointCloud 1.0, Python 2.7, PDAL 1.1, Numpy 1.10 and Scipy 0.17. We stress that all the facts should be indicative at best, because dealing with massive data introduces a strong hardware factor. For instance the same loading method (parallelized PDAL) used in the reference benchmark of van Oosterom et al., 2015 is 4 times slower with our hardware. Moreover, the PCS uses several layer of caching whose influences may blur the results.

RESULT AT THE SYSTEM LEVEL Overall, we load several Billion points into the PCS, perform several processing in and out of base (second to hour), extensively use simple and complex filtering (response time from millisecond to second), convert points to images and vectors, and output points ($\geqslant$ 100k pts/s). The entire system works as intended and is efficient and powerful enough to be used in research settings.

DATA SETS USED For this work, we mainly use four data sets (including (IQmulus, 2014)) illustrated in Figure 31. They were chosen to be as different as possible (size from Million to Billion, sensing from active to passive, wavelength from few nanometre to Near Infrared, nature of sensing from surface to volumetric, different number of attributes from none to 21) to further evaluate how proposed methods can generalise on different data. We emphasize that the Vosges data set is a massive aerial Lidar point cloud covering mountains and forests.

HARDWARE We tested all our methods on two settings corresponding to two target users. The first use case is non-specialised user with non-dedicated hardware. To this

Table 1: Figures of the Point cloud data sets used in experiments.

data set	Type	Nb. of points	Nb. of original files	Spatial coverage	Nb. of attributes	Typical spacing
Vosges	Aerial Lidar	5.2B	~1450	1330 km^2	9	1 m
Paris	Terrestrial Lidar	2.15B	~ 750	42 km	21	1 cm
Stereo	Stereovision	70M	16	3 m^2	6	0.1 mm
Medical Imaging	Medical Imaging	1.95 M	1	20 dm^3	0	0.6 mm

end the setting is simple and portable (the point cloud server is hosted on a virtual box on an external drive). The second use case is for specialised user, with dedicated hardware. The setting is powerful and offers much more storage place (dedicated server with 12 cores, 20 GB RAM, SSD for OS, regular disk for storage, Ubuntu 12.04). Timing are indicative because of influence of caching and configuring.

Storing groups of points in a RDBMS

Table 2: Creating and indexing patches for the test data set . *: estimation. Grp: Grouping

data set	Grp Size	Grp Dim	Patch nb (k)	Avg pts/patch (kpts)	Patch index size(MB)
Vosges	50m	2D	580	8.9	27+15
Paris	1m	3D	6570	0.325	300+150
Stereo	$\frac{1}{250}$m	2D	180	0.4	12+3
Medical	$\frac{1}{100}$m	3D	4.8	0.4	0.4

Table 2 gives results on the grouping and indexing aspects.

POINT GROUPING STRATEGY Points must be categorised into groups that will make sense for subsequent uses of the point cloud. Groups of points must be big enough so the number of rows is tractable, but not too big because getting only one point still necessitates obtaining the entire group. We designed these grouping rules with two criterias. First the number of rows is less than a few millions so that the index fits in the server memory (Table 2). Second the range of number of rows is still manageable for classical GIS software (e.g. QGIS). We can afford to have arbitrary large groups as a result of the PostgreSQL TOAST[8] storage system.
Grouping is done at data loading, but we can change the groups and grouping rules at any time. Index creation is very fast (a few seconds to a few minutes), and the index size is $\leqslant 1$ % of the point cloud size.

STORING PATCHES AND NOT POINTS Indexes are built on patches and not on points, and thus are several orders of magnitude smaller and much faster to build. We estimate the size of indexes if we were to store one point per row rather than one patch per row.

8 http://www.postgresql.org/docs/current/static/storage-toast.html

Table 3: Analysing compression ratio and compression/decompression speed.

data set	Points	Disk size	Server size	Compression ratio	Comp. Speed M pts/s	Decomp. Speed M pts/s
Vosges	5.2B	170 GB	39 GB	4.36	4.49	4.67
Paris	2.15B	166 GB	58 GB	2.86	1.11	2.62
Stereo	70M	1 GB	490 MB	2.05	2.44	7.38
Medical	2M	23 MB	7.7 MB	2.98	7.66	25.8

For this we measured how PostgreSQL index size and build time evolves with the row number. The results are mostly linear (tested from 10 kpoints to 10 Mpoints), as seen in table .

XYZ points table size	RTree index size	RTree index building time
65 MB/Mpts	52 MB/Mpts	18 s/Mpts

Using this scaling behaviour, we estimated the spatial index size if the point were not stored by groups but individually (one point per row). Without surprise index size and built time would become intractable if points where stored one by one and not by groups.

data set	Estimated point index size (GB)	Estimated point index build time (h)
Vosges	2600	290
Paris	1000	120
Stereo	35	4
Medical	132	1 min

COMPRESSING POINT CLOUDS Patches are compressed before storage using the dimensional pgpointcloud compression[9]. We compare loaded data set space occupation on the server with original binary files on the disk. In our case, patches are compressed attribute-wise, with either a run-length, common bit removal, or zip strategy. This means that for each patch, each attribute (dimension) is compressed independently using the strategy which is deemed optimal for this attribute. The compressing efficiency widely varies depending on the data and the type of points attributes (See Table ??). As a comparison, a tool specialised on .las file compression like the one proposed by (Isenburg, 2013) achieves a *8.3* compression ratio on the Vosges data set. It uses a more adapted delta encoding approach for XYZ and time and does not compress the attributes.

Compressing and decompressing data introduces an overhead on the data access. We estimate it by profiling the uncompress and compress functions. Again, the overhead is dependent on the type and number of attributes. For instance stereo contains double attributes that are compressed with the zip strategy, which is slower in compression. See Table ?? for the result.

9 https://github.com/pgpointcloud/pointcloud#compressions

SPATIAL OR TEMPORAL GROUPING In this experiment we use two different grouping methods on terrestrial Lidar data. This type of Lidar (Velodyn) rotates around Z axis at 10Hz (see Figure 21 58 for one rotation), and is commonly used to perform object tracking (see the work of Azim and Aycard, 2012 for instance).
In such a case, the main filtering aspect may be temporal, and not only spatial. In the temporal scenario, we group points acquired every 4 ms together, and display the convex hull for easier visual understanding. In the spatial scenario, we group the points with a 1m grid.
Without surprise, temporally-grouped patches have a more regular number of points, whereas the spatially-grouped patches have a much more diverse density (see the histogram of Fig. 32). In both cases the compression is similar, as the filtering time.

In the Spatial grouping, knowing precisely the sampling rate (10Hz),it is then easy to get points that are sensed during a turn n, but not before or after. This capacity would be the basis of object/change detection. .

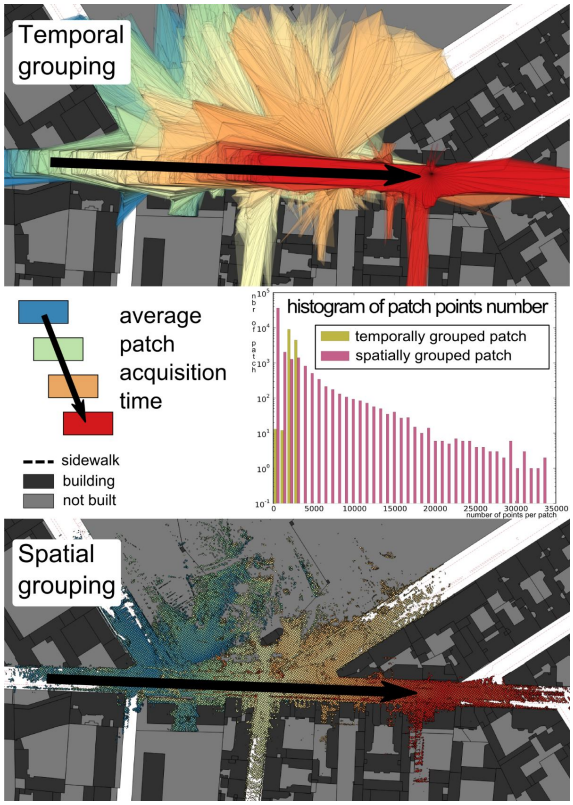


Figure 32: Temporal (left) and spatial (right) grouping on velodyn terrestrial Lidar data, with histogram of number of points per patch.

ADAPT PATCH GROUPING RULES In our solution, points are grouped at loading time with fixed rules. This system is well adapted to point clouds with homogeneous density, like aerial Lidar. However, these fixed grouping rules are not optimal for terrestrial Lidar, where the point density varies strongly based on the distance to the Lidar device.

We propose a mechanism to change the patch grouping rules on loaded data sets. The user fixes a target patch density depending on the expected number of rows and

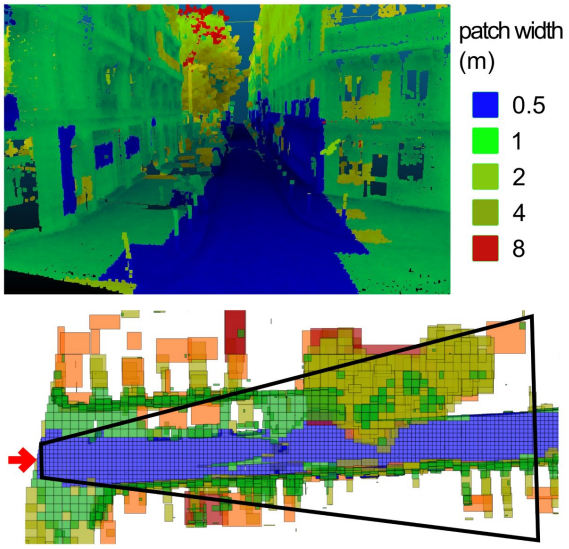


Figure 33: Illustration of variable patch size repartition in an urban point cloud.

expected usage of the point cloud. In the example, we target a density between 0.5 and 2 kpts /m^3. We then iteratively split/merge patches to try to meet this target. Figure 33 illustrates in 2D and 3D views of patches of different size but containing roughly the same number of points. With the proposed targets, the global number of patches is roughly the same, with the benefits of having less frequently too small or too big patches, which particularly shows in the histogram of the points per patch for fixed size and variable size patches (34).

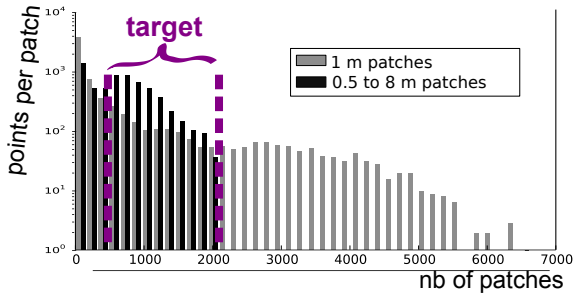


Figure 34: Histogram of points per patch for constant and variable size patch. Using variable size patch strongly reduces the number of very small or very large patches. Total number of patches is roughly conserved.

This adaptive grouping size also reduces the global quantization error.

CHOOSING GROUPING RULE Overall, the chosen grouping rule has no impact on performances as long as the number of rows remains in the same magnitude (few millions). The impact of using different grouping rules is essentially to enable different applications. For instance the adaptative grouping rule produces patches with a much more constant density, which is useful for many processing methods. We refer the reader to Cura, Perret, and Paparoditis, 2016 (working paper) for more details on point cloud density correction.

PARALLEL LOADING ('SERVER ORIENTED') In one night, we aim at loading the data sensed by a Lidar system during one day. Indeed, a mobile mapping vehicle may be acquiring data all day long, but would most likely not operate during the night (street views would be useless). Thus the data acquired during the day must be dealt with during the night so as to keep up when vehicle is used everyday.
Please note that this requirement is specific to only one of our four data sets.

The points are stored in files, over a gigabyte network. We use a specialised program to convert the points file into ASCII values (CSV). For .ply point files the program is a modified version of the RPly library[10], for the .las files the program is LAStools [11]. The points in ASCII values are streamed to a 'psql' process that is connected to the database. The 'psql' executes a 'COPY' SQL command that reads the ASCII streams and creates and fills a table with the values from the ASCII stream. When the file has been fully streamed, we use a SQL query to create points from attributes and group them into (compressed) patches. These patches are inserted into the final patch table. This pipeline (Figure 24, page 60) is executed several times in parallel, each pipeline working on a different file. The process is pipped so there are no intermediary files written to disk.

DISTRIBUTED PARALLEL LOADING ('CLIENT ORIENTED') In this experiment, we use clients to send uncompressed patches to the server . The clients read point cloud files (.ply in our experiment, using the plyfile[12] Python module). Then, each client groups the points into patches using a custom Python module. The patches are sent to the server through Python. The server compresses these patches and adds them to the final point cloud table. This experiment is a proof of concept; therefore, we limit the number of clients to one computer, using seven threads.

RESULT We load "Vosges" and "Paris" data sets through 'Parallel loading', and "Stereo" through Distributed parallel loading' (Table 4). Morevover we load a part of Vosges data set using PDAL (multi-threaded) to enable comparison of our result with van Oosterom et al., 2015. We estimate the loading speed of the Vosges data set with PDAL at 300kpts/s, and the writing speed at about 750kpts/s.

We try to roughly estimate the bottleneck of each method in the following way: we vary ne number of core used. If the timing is linear with the number of core, the process is CPU-bound, that is CPU is the bottleneck. Else, input/output (I/O) is the most likely bottleneck.

For PDAL, the bottleneck is clearly the CPU, for both of our methods, the input/output (I/O) may also play a role. Indeed, the point files are read over the network, and the point tables are stored on the SSD, but the final patch table is stored on the regular disk, which also limits how many threads can write data on it at the same time (we observe almost linear scaling for all methods up to 7 threads).

10 http://w3.impa.br/~diego/software/rply
11 http://lastools.com
12 www.github.com/dranjan/python-plyfile

Table 4: Loading and writing time for each point cloud data set, using various methods.

data set	Loading time	Parallelism	Loading speed kpts/s	Writting speed Mpts/s	writting limitation
Vosges	11h30	8	125	1.1	write speed
Paris	8h	6	74.5	0.2	read / uncompress
Stereo	7'20	7	160	0.55	read

Please note that PDAL and our methods do not use the same grouping rules (PDAL uses fixed max size (streaming friendly), while we group points into fixed size cubes (necessitate to read the whole input file before grouping)). Results are in the table 4.

Point Clouds and Context

First, we demonstrate the construction of several two-dimensional (2D) vectorial visualisations of point clouds. The PCS can work on all point clouds at the same time, transparently for the user. Point clouds can also be used conjointly with other GIS data (raster and vector). Lastly, we show an example that demonstrate the use of the sensor trajectory metadata (Sec. 2.4.3.5 p. 77), and the creation of graph / topology at the patch level.

Coverage visualisation

Creating a coverage visualisation is easy (about 30 SQL lines) and fast (about 150s, one thread) with our point cloud server.

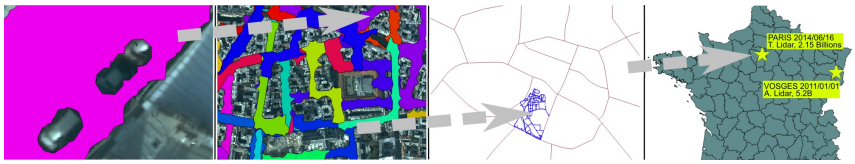


Figure 35: Successive visualisations (various scale) of point cloud coverage, see 2.4.3.1 for details.

Indeed, instead of working with billions of points, we can work with millions of patches (generalising the points).

We created several visualisations for the Paris data set, ranging from 5MByte to 100kByte, each adapted to a different scale and purpose. (see Figure 35)

- 1:25 to 1:1500: Precise, occlusions visible (~1m).
- 1:1.5k to 1:15k: Understand acquisition structure and road morphology (~8m).
- 1:15k to 1:200k: Use the trajectory. If not available, fabricate a trajectory-ersatz through basic straight skeleton.
- $\geqslant$1:200k: A simple point with text attributes for details, linked to a relational model.

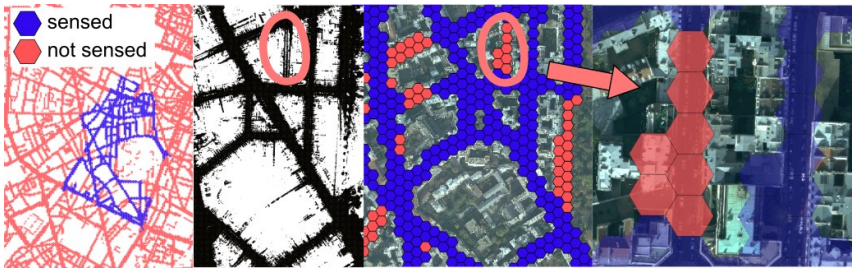


Figure 36: "To-Do" hexagonal map showing the places where the mapping vehicle has not sensed enough points (red hexagon), and where the sensing is sufficient (blue hexagon). Without this map, the zoomed missing part is challenging to notice on raw data.

As a proof of concept, we propose a coverage hexagonal grid (see Figure 36), conceptually identical to regular grid, with some benefits (see Sahr, 2011). The idea is to visualise both what was sensed and what remains to be sensed in a given area (here whole Paris city), to help plan data-sensing missions. The whole process is fully automatic. We fabricate an hexagonal grid over the extent of Paris (about 30s), and remove the hexagons that are in buildings or too far from road (about 60s). Then we colour the hexagons depending on whether the point cloud really covers it or not(about 30s). Such a visualisation is easy to create (about 30 minutes of design), and could be tailored to more specific needs.

Using several point clouds

As a proof of concept of integration of several point clouds, we demonstrate the conjoint use of stereo-vision point cloud and Lidar point cloud. For this experiment, we choose to use PostgreSQL inheritance mechanism. The idea is to create a 'parent' table. We set the Lidar table and stereo-vision table to be a 'child' of the 'parent' table. Then we can query the 'parent' table as if it was a super-point cloud comprised of all the others. Querying the 'parent' point cloud is as fast as querying one point cloud, and we correctly attain points from both point clouds. We stress however that all the child point clouds index are used, which would limit the scaling of this method.

Conjoint use with other GIS data

We commonly used vector data with point clouds for various research projects. Here, the scenario is that we have an accurate but slow pedestrian crossing reconstructing process. We want to reconstruct the pedestrian crossing at a given intersection, so we use advanced filtering to provide the appropriate points to the reconstructing process (see Figure 28). To demonstrate the possibilities, we use the following:

- Corrected version of ODParis[13] building layer (350 k rows),
- Lidar sensor trajectory (42 k points regrouped in 900 rows),
- Road network data of BDTopo[14] (32 k rows),
- Aerial image in a PostGIS raster table (110k rows, each 30×30 pixels (10 cm)).

We filter Paris point cloud to obtain patches :

13 http://opendata.paris.fr/page/home
14 http://professionnels.ign.fr/bdtopo

Table 5: Result of filtering.

Total points	Found points	Filtering no rast.	Filtering w. rast.	Filtering optim
2.15 Bpts	1.2 Mpts	~30ms	~ 5s	~30ms

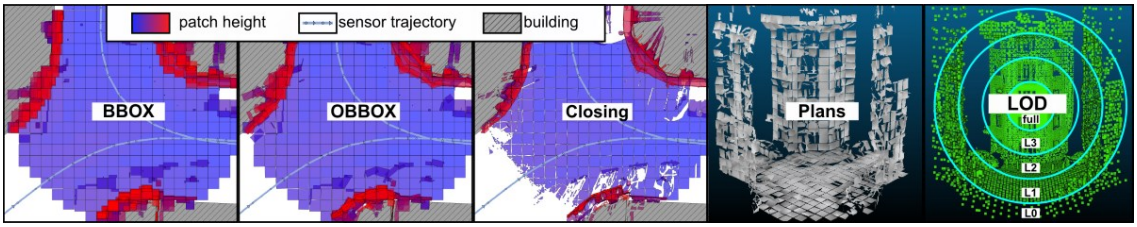


Figure 37: Streets and buildings. Generalisation like Bounding Box, Oriented BBox, Spatial closing. Closing on 3D plan detection. Level Of Details.

- near street 'Palatine' and 'Servandoni' ($\leqslant$ 10 m+ road width),
- near Lidar acquisition centre trajectory ($\leqslant$ 3 m),
- far from buildings ($\geqslant$ 1 m),
- with high density ($\geqslant$ 1000 points /m^3),
- where the aerial image has a colour compatible with street. markings (240 $\leqslant$ mean intensity $\leqslant$ 350)

The point cloud server finds all the patches concerned in about 0.6s (with index and optimally written query) (see Figure 28 and Table 5).

Point cloud as a raster or vector

We construct abstract representations of patches that are sufficient for one task, and are much more efficient than using the points, including the following:

- 2D bounding box ('bbox') (default)
- oriented bounding box ('obbox'), light
- multi-polygon obtained by successive dilatation and erosion of points ('closing'), big to store, very accurate

These generalisations are about 0.5% of the compressed patch size. We also tested 3D generalisations, either by extracting primitives (plan, closing on 3D plan, cylinder) or using LOD.

Lafarge et al., 2013 showed that the urban point clouds can be accurately represented by primitives. For instance, a dozen plans accurately explains (distance $\leqslant$ 1cm) 70 % of this scene (Figure 37). We extensively tested an orthogonal approach, where instead of making a new object to generalise a group of points, we represent it by a subset of well chosen points of this group. The method and its applications (adaptive LOD, density analysis, and classification using density features) are explained in Cura, Perret, and Paparoditis, 2016 (Working paper).

Using trajectories with point clouds

We imported the Paris trajectory data (the successive positions of the Lidar sensor every few ms). In fact, using a constrained data model resulted in discovering errors in the raw trajectory data (some rows were corrupted). Trajectories can be used for filtering point clouds (for instance, Sec 2.4.3.3).
We demonstrate the use of trajectories for processing in the following scenario. The goal is to localise all the pedestrian crossings of the Paris data set (few minutes). We (conceptually) walk along the trajectories, and every three metres we retrieve the patches closest to the trajectory. We use a crude marking-detection function on these patches (percent of points in given intensity range). By thresholding this score, we can be conservative or very selective (i.e. favour recall or precision). Recall is the amount of correctly found crossing over the total number of crossing. Precision is the amount of crossing that were correctly found divided by the total number of found crossing.

For instance, with a recall of 0.95, we have a precision of 0.5, and we already filtered the point cloud by a 4.8 factor. This indicates that we reduced the number of points to consider by a factor 4.8 at the price of dropping 5 % of the pedestrian crossing to be found. This could directly be used as a prefiltering step for a more costly pedestrian crossing detector, which would work on 4.8 times less points (at the price of missing at least 5 % of pedestrian crossings).

Orthogonally, with a recall of 0.16 we have a precision of 1, filtering the point cloud with a factor 100. This indicates that we can guarantee that the found pedestrian crossing are effectively pedestrian crossing for 16% of those. This could be useful for fast prototyping. Indeed we may want to test a more subtle pedestrian crossing detector. In this case knowing there is no false positive is important to evaluate the new method.

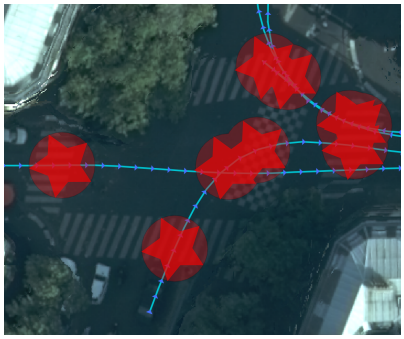


Figure 38: Rough pedestrian crossing detector.

Point cloud as Graph / Topology

We generate a graph embedded in 3D from patches and propose three examples of applications. First we generate the graph by creating a node per patch, the node being at the patch centroid. We de-duplicate results to correctly deal with the fact that the acquisition vehicle made several passes at this place. Patches are regularly spatially placed (for instance forming cubes of 1 m^3). Then we construct an edge for each pair of nodes that are spatially close. Depending on the threshold, we can use 4,8 or more connectivity (the graph is in 3D). Figure 39 illustrate the different kind of connectivity.

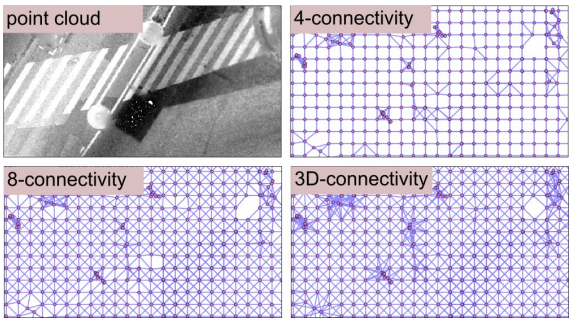


Figure 39: Example of possible connectivity by connecting patch centres that are closer (3D distance) than a threshold (1.4, 1.5, 1.9 metres).

The edge weight is the 3D geometric distance between the nodes (or a more complex measure).

SHORTEST PATH The first example takes advantage of geodesic distance to compute the shortest path between two groups of points (see figure 40). We construct two graphs, one for the regular fixed-size-grouping (1 m^3), and one for a variable-size-grouping (0.25 to 1 m wide patches). We use PGRouting to find the shortest path (about 0.1 sboth cases). Both results are similar, with the shortest path along varying size patches being a better approximation, as expected. This functionality of geodesic distance could typically be used by other advanced processing methods.

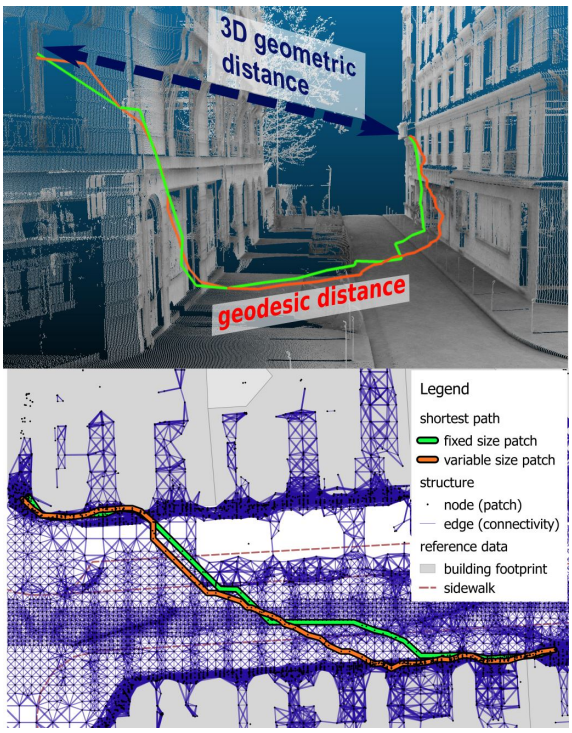


Figure 40: Two views of shortest path on regular (yellow) and varying (orange) size patches.~ 100ms

SEMANTIC ISOCHRONE The second example is intended for semantic point clouds. We suppose that each patch has already a rough classification available (aggregated from available point classification or the result of a direct patch classification(Cura, Perret, and Paparoditis, 2016, Working Paper). We integrate this classification with the geometric distance to create a semantic-geometric distance. We use the geometric distance divided by a measure of similarity of the patch classes. For instance two patches are spatial neighbours, one being a ground patch and another a building patch. The semantic-geometric distance will be large.

The graph can then be used for assisted selection. In this scenario we would like to get all the points pertaining to a façade. A user selects one patch on the façade, then all the patches within a given semantic-geodesic distances are selected (red/yellow) using PGRouting isochrone functionality. This results in selecting only the given façade, as opposed to using a simple geometric distance which would also select point on other façades (geometrically close, but not connected see Figure 41).

More generally high level object reconstruction algorithm would need this kind of feature.

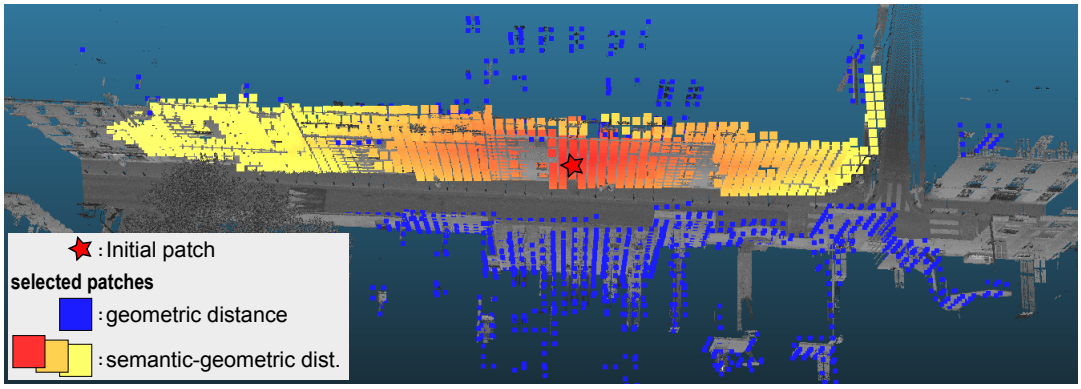


Figure 41: Isochrone from the red to yellow, blue points are selected when using geometric distance, but not selected when using semantic-geodesic distance (about 0.4 s).

ROAD NETWORK RECONSTRUCTION The third example use a simplified graph as the starting point to reconstruct a road network (See Figure 42). The idea is to regroup nodes of the graph to simplify it. We generate the simplified graph by sampling patch centroid on a voxelic grid (8 m) coarser than the typical patch size (1 m wide), taking in priority the patch with greatest number of points, and removing patch that are not flat enough to be on the ground. We could directly use semantic information if it was available to keep only ground patches. Edges are created as usual (geometric proximity).

Please note that such down-sampling of patches is easy to write and fast, and could as easily replaced by more subtle aggregates :

```
SELECT DISTINCT ON (floor(X), floor(Y), floor(Z))
  patch_centroid AS node
FROM patch_table /*X,Y,Z are patch centroid coordinates*/
WHERE /*trying to characterise ground patches to avoid getting facade and
    trees patches*/
  num_points > 1000
  AND patch_height < 0.4 --in meter ...
ORDER BY floor(X), floor(Y), floor(Z)
```

```
, num_points DESC -- large patches have priority
```

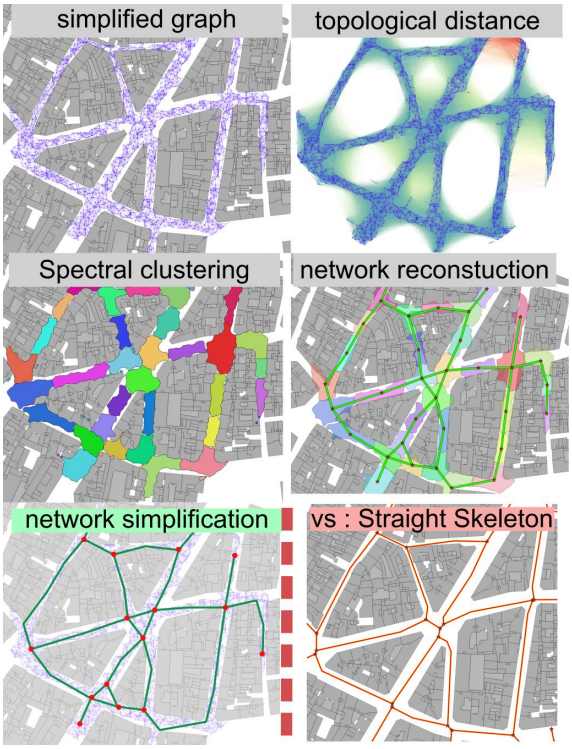


Figure 42: From simplified graph, all topological distance from a vertex to any vertices close enough are computed with PGRouting, the result is clustered using spectral clustering of Pedregosa et al., 2011. We then reconstruct the network with PostGIS Topology, and perform a final simplification step. For comparison an automated result using a straight skeleton based method.

Using PGRouting and PostGIS, we compute the accumulated graph-distance between all the pairs of nodes close enough (50 m). We can see those $(\texttt{node1}, \texttt{node2}, \texttt{t\_distance})$ as edges of another graph. We create the sparse affinity matrix of this graph using Networkx (Hagberg, Schult, and Swart, 2008). Then we use spectral clustering with Scikit-Learn (Pedregosa et al., 2011), each node is then attributed to a cluster. Networkx ans Scikit-Learn are python modules that we use in base via plpython. We can replace the cluster by their centroid, and build a network by computing cluster adjacency relations. We then simplify this network by "healing" 2-connected edges. We compare this result with a more traditional straight-skeleton based approach (use morphological operation to produce a polygon with holes representing the surface of the streets, use straight skeleton to produce centrelines, clean the straight skeleton result with morphological operations, build a network, topologically clean the network).

Point Cloud Filtering

FILTERING OVERVIEW Overall, filtering of patches is very fast on the Point Cloud Server when an index is used ($\leqslant$ 0.1s). We tried a great variety of combination of filtering conditions, and always observed this kind of timing provided that indexes were used. The Section 2.4.3.3 was designed to give an overview of filtering conditions.

Advance filtering condition examples can be found in the method wiki[15]. Finding the patches is almost always much faster than actually retrieving them.

Because of caching and the influence of how the query is written, figures are only indicative. Filtering patches using the indexed functions takes about 0.01s, even when using many conditions at the same time. This includes filtering with spatial (2D and 2D+Z), temporal, any other attributes, density, volume, etc. This also includes using vector generalisation. Filtering with other GIS data (vector) is slower (10s), except when special care is taken to optimise the query (0.01ms). This includes using distance to other vector layers, using other vector layer attributes (e.g. height of building), using time associated with vectors, etc. Lastly, very complex filtering may take from 10s up to several minutes depending on the number of patches concerned.

Exporting

In this section we list the results for the various output methods of the PCS.

We estimate the output speed using parallelised (8 threads) PDAL to 750kpoints/s on the Vosges data set, by simply measuring the time taken to output few hundred million points.

USING POSTGRESQL DRIVERS / CONNECTOR ('CLIENT ORIENTED') We create a Python method that works on a client computer. It reads uncompressed patches from the server and directly writes them to disk (saving it as Numpy double array). Using seven parallel workers, the result is in Table 4 p. 74.

PLY FILE AS A SERVICE (PLYFAS) 'SERVER ORIENTED' We create a service that writes ASCII .ply files at a given network place. The functions (API) have options to perform all kinds of filtering. We exported several files from the Paris data set, with various filtering options and LOD (from Cura, Perret, and Paparoditis, 2016, working paper). The global output time observed is around 15 k points/s per worker with a scaling of up to seven workers.

LENS FOR TRADITIONAL GIS Points are in fact a PostgreSQL materialised view that store points that are defined by the lens spatial extent and attributes. We also add a trigger on the lens table so to refresh the point view upon changes on lens. Optionally, a QGIS plugin[16] can also be used to improve interactivity (instantly autosaving changes concerning PostGIS layers). If the lens is small enough, this method is interactive (~ 2s). See figure 30 page 66.

STREAMING TO BROWSER 'SERVER ORIENTED' We performed a test of point cloud streaming to a WebGL application, using a Node.js server as the 'man in the middle'.

- The browser is set to a geographical position, and then requests the points around this position to the Node.js server.
- The Node.js server connects to the PCS to request the points.

15 https://github.com/Remi-C/Pointcloud_in_db/wiki/Point-Cloud-File-As-A-Service#using-custom-codes

16 http://remi-c.github.io/interactive_map_tracking

- The PCS uses indexes to find patches and extract points that are then streamed to Node.js server through cursor use.
- The Node.js server compresses the point stream and sends it to the web browser.
- The web browser parses the stream, puts the points into buffers, sends the points to the graphic card and display them through shaders (WebGL).

We observed a reduced throughput (~ 20 kpts/s, monothread) because data is inefficiently transmitted as text, and is serialised/deserialised multiple times.

Processing Point Cloud with the Server

The PCS can be extended by in-base and out-of-base processing. In-base processing are methods that are executed by the database from within, whereas out-of-base processing are regular processes executed outside of the database (possibly on other computers) that get data from database, process, and then write results in database or elsewhere.

We demonstrate how easy it is to create new in-base processing methods. As such the methods are only cited to illustrate these capacities. Some details may be found in Cura, 2014b.

IN-BASE PROCESSING Fast prototyping is vital for wider point cloud use. We demonstrate the potential of using high level languages within the database to write simple processing methods. The experiment is not to create state-of-the-art processing methods, but to measure what a Python/R beginner can do in two weeks (designing methods and implementing), using well established tools like Scikit-learn (Pedregosa et al., 2011) or the PointCloud Library (Rusu and Cousins, 2011).

((P): directly working on patches, can be used out of the box on all point clouds; (R): working on rasterised point clouds, need to use a point cloud to raster conversion method first (in base or out of base))

- (P) clustering points using Minimum Spanning Tree
- (P) clustering points with DBSCAN (Ester et al., 1996)
- (P) extracting primitives (plans and cylinder)
- (P) extracting a verticality index (using Independent Component Analysis)
- (R) detecting façade footprint
- (R) detecting cornerstones
- (R) detecting road markings

We also used the server for complex out-of-base processing (classifications), although for the sake of brevity this is detailed in a standalone article (Cura, Perret, and Paparoditis, 2016).

Storing groups of points in a RDBMS

We introduce storing groups of points in Section 2.3.1 on page 57, and we study different grouping rules in Section 2.4.1 on page 69.
Storing groups of points in the PCS offers strong advantages in term of compression, indexing and generalisation. Yet it all depends on the hypothesis that points are grouped into groups that are meaningful for the intended applications. Both spatial and temporal grouping produces good results. Spatial grouping with fixed size patch can be a problem when the density varies much (terrestrial Lidar), as patch may contain very few or too much points. We experimented with varying patch size and demonstrated that resulting patches have a much more regular density.

We noted a limitation concerning the point cloud types which are strongly constrained, thus adding or removing attributes is not immediate. As a perspective, an inheritance scheme between point types would solve this problem.

We demonstrated that storing groups of points is well adapted to store billions of points per table. Yet the PCS would have trouble going over a few thousand tables, theoretically limiting the total number of points to the 10 trillion-points range. To go beyond that, we would need to use supplementary PostgreSQL sharding and clustering capabilities. Those capabilities exist but have not been used yet for point cloud, to the best of our knowledge.

Storing groups also enables a generalisation approach, which may have the potential to accelerate and facilitate many point cloud usages. In this work we only considered a few generalisations, and used them in limited ways. Much more advanced generalisations and usages would be possible (for instance, using Gaussian Mixture).

Loading

We present several methods to load points into the PCS (Sec. 2.3.2 on page 59), and test them on several datasets (Sec. 2.4.2 on page 73). We sucessfully demonstrate a sufficient speed to fulfil our practical requirement of loading one day of sensing data in less than one night. Examining (Martinez-Rubi et al., 2014, Table 2) shows that data loading could be much faster. In the Parallel loading ('Server oriented') scenario, points are converted to ascii and streamed, which is a wast of ressources. The PCS could directly read .ply or .las files. In this scenario the database performs the grouping via generic SQL query. It might be faster to create a tuned C function to do this.

In the Distributed parallel loading ('Client oriented'), the client performs the grouping, but the database still performs the compression. The client could also compress, saving bandwidth and computing time for the server. Moreover, our prototype is written in Python and could be written for efficiency in lower level languages. As such the relatively recent initiative, PDAL[17] has gained maturity, and would be the ideal candidate to solve these two limitations.

In a more distant perspectives, we could skip reading post processed .las or .ply files, and directly read the raw sensor data, which might be nevertheless difficult due to current lack of driver and standard accessors.

17 http://www.pdal.io/

Point Cloud and Context

Point clouds are not only set of points and also contains very important meta-data (Sec. 2.3.3 on page 60). We used these meta-data in several ways (Sec. 2.4.3 on page 74).

We demonstrated that such meta-data can be useful to create multi-scale visualisation of point clouds coverage, as well as help to analyse sensed area ("Todo" map).

Each point cloud meta-data scheme must defined and enforced by the user, making it hard to share. A standard minimal data model would be necessary to facilitate exchanges, similar in spirit to the INSPIRE[18] European directive.

A shared meta-data scheme allows to use several point cloud together. We tested the PostgreSQL inheritance mechanism so all point cloud are parts of one meta-point cloud. Current limitations of this mechanism would prevent it to be used on more than a dozen point clouds, but perspectives exist to solve this problem (using rule system or enforcing a table-wide pre-filter based on table coverage).

In the PCS the point clouds also have representations compatible with other GIS data, such as vectors and rasters. Conjointly using vectors, rasters and point clouds offers a new world of possibilities. We face data fusion issues, like difference in precision, generalisation, fuzziness, etc. Moreover, vector, raster and point cloud data may be acquired at different dates.

In-base conversion from point cloud to raster are currently very slow and tailored, being based on SQL queries. A python-based prototype[19] method may solve these limitations.

Going one step further, we demonstrate that point clouds could be generalised as graph, which opens new possibilities, such as graph based distance, semantic selection and road network reconstruction. However, the current approaches build the graph using plain PostGIS SQL queries that can not scale well beyond the million of patches. The bottleneck is simply the conversion from patches centroid to a graph based on adjacency, and could be done directly using powerful specialised library, such as Boost graph library[20]. Moreover, the reconstructed road network by either methods has large improvement margins (topologically and geometrically). It would be possible to mix PostGIS Topology (2D partition of the space) for graph queries.

Filtering point clouds

The PCS has very advanced capabilities to access a subset of the point clouds (Filtering, Sec. 2.3.4 on page 63). Filtering relies on indexing and we demonstrate it is very fast provided the filtering conditions are indexed and the points are grouped in meaningful groups (2.4.4 on page 80). Level Of Detail and metadata (trajectory) naturally integrates well into the filtering conditions.

The filtering conditions are especially useful when using other GIS data. It would be possible to go much further towards complex filtering, by performing algebra between several rasters, using attributes of vectors to filter patches, etc.
Our entire strategy relies on filtering patches first, then filtering points. In cases when the patch filtering condition does not filter much, the system is useless.

18 http://inspire.ec.europa.eu/
19 https://github.com/Remi-C/PPPP_utilities/blob/master/pointcloud/patch_to_raster.py
20 www.boost.org/doc/libs/1_58_0/libs/graph/doc/

Exporting

The PCS being based on a popular RDBMS, many ways exist to access data stored in it (Sec. 2.3.5 on page 64). We demonstrated an array of export methods (Sec. 2.4.5 on page 81), from classic server based export to multi-client based, up to the notion of point clouds as service.

The point cloud server can output data in many ways and thus be easily integrated into any work-flow. We, however, feel that the current speed (100 k points /s, around 2 MByte /s) is too low. It could be easily accelerated using binary outputs and by decompressing patches directly on clients.
Similarly, the lens feature is limited, adapted LOD could be chosen automatically, but this involves modifying the GIS used for visualisation. Perhaps the true evolution of the point cloud server would be to stop delivering points, and instead deliver a service that could be queried through standard mechanisms. For instance, the transactional Web Feature Service (WFS-t) format could be used to send points out of the box, simply using a geo-server between the client and the point cloud server. This could be a revolution in point cloud availability, similar to what happened to geo-raster data (e.g., google map WFS).

Processing Point Cloud with the Server

One of the advantages of using the PCS is the opportunity to not only store point clouds, but also the methods to process it (Sec. 2.3.6 on page 67). We demonstrated the PCS capabilities for fast prototyping of in-base processing methods (Sec. 2.4.6 on page 82).

The methods we designed are proof of concept, and far from real state of the art. In-base processing offers many opportunities because it is close to the data and can be written with many programming languages. Yet, it is also intricately limited to one thread and the amount of memory allowed for PostgreSQL. The execution is also within one transaction. It may also be hard to control the execution-flow, during the execution. However, the Python access both from within and outside the database shows the possibility to write more ambitious processing methods with several parts executing in parallel as well as communicating, dealing properly with errors, etc. We sucessfully integrated the PCS into a more complex classification framework in Cura, Perret, and Paparoditis, 2016.

Future work

Patches and their generalisations are perfect candidates to perform fast and efficient registration (cloud-to-cloud, cloud-to-raster, etc.). (See figure 43) Indeed, the classical solution for registering two point clouds relies on a lot of point to point distance computing (in Iterative Closest Point for instance, Besl and McKay, 1992). With large point clouds the problem grows intractable, and thus a common solution is to subsample the point clouds to reduce the number of points. This introduces errors, and is still less than perfect. Using extracted primitives would be better, as visually explained in Figure 43. Indeed higher level primitives contains much more information (more abstract), and are much less numerous (more synthetic), both being great for faster registration.

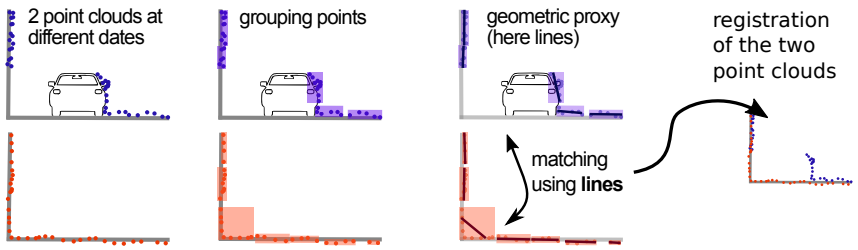


Figure 43: A schematic example of the benefits of using generalisation of points for fast registering. The critical part of matching could be done on geometric proxies instead of points, reducing the number of entities to be matched by a factor of at least 10^3.

Having all the meta-data, the trajectory (or camera position matrices), and the raw data, it would be possible to change the trajectory (matrices) and regenerate the point cloud with updated coordinates, all of this from within the database. Indeed the trajectory or camera position are usually known up to a positioning error, being the result of a process (Structure from motion, GPS positioning, etc.). Yet those positions could be improved by exploiting other data, manual correction, etc. In this case, the improved trajectory/ positions could be used to re-generate the point cloud, leading to more accurate point clouds and limiting data duplication. Processing of point clouds would extract landmarks, which could be matched with a landmark database.

CONCLUSION

In this article, we presented a complete point cloud server system based on groups of points (patches). Using these patches as generalisations, we propose solutions for all point cloud basic user needs (loading, storing, filtering, exporting and processing). The system is fully open source and thus easily extensible and customisable using many programming languages (C, C++, Python, R, etc.). Our system opens new possibilities because of intricate synergy with other geo-spatial data. Lastly, we proved through real-life uses that this system works with various point cloud types (Lidar, stereo-vision), not only for storing point clouds, but also for processing. As perspective, we could explore in-base re-registration from trajectory and raw data, in-base cloud-to-cloud registration, in-base classification, and point streaming, as well as scaling to thousands of billions of points.

3

STREETGEN : PROCEDURAL MODELLING OF STREETS

The goal of this thesis is to reconstruct streets so that the resulting street modelling can be used in many applications (visualisation, traffic simulation, spatial analysis, etc.). Reconstructing an object always requires to somehow have a model of it (although it can be implicit). Because we target several applications, we need a modelling that could be easily extended and accessed, and be flexible with great expressibility for what it can model. Indeed, street modelling requires to model street objects, street surfaces, but also several topologies for traffic. To this end, we create and store the street modelling in a database server. This way, street modelling and sensing data are in the same place, and this allows scaling and multi-users usage. This street modelling is used in all other parts of this thesis.

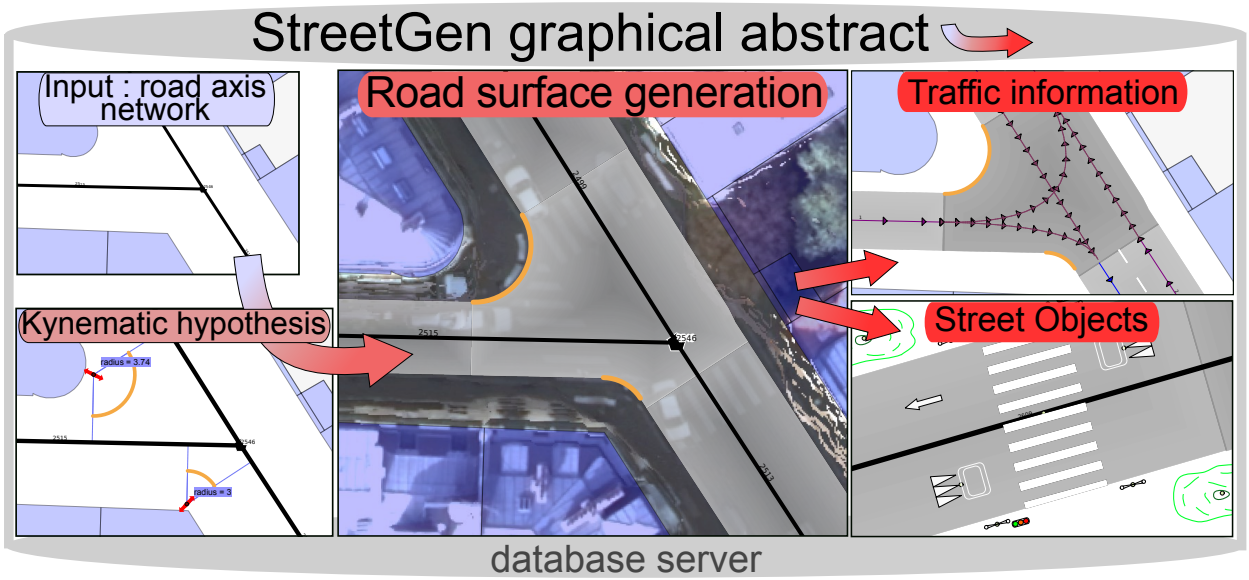


Figure 44: StreetGen graphical Abstract.

Streets are large, diverse, and used for several (and possibly conflicting) transport modalities as well as social and cultural activities. Proper planning is essential and requires data. Manually fabricating data that represent streets (street reconstruction) is error-prone and time consuming. Automatising street reconstruction is a challenge because of the diversity, size, and scale of the details (~ cm for cornerstone) required. The state-of-the-art focuses on roads (no context, no urban features) and is strongly determined by each application (simulation, visualisation, planning). We propose a unified framework that works on real Geographic Information System (GIS) data and uses a strong, yet simple hypothesis when possible to coherently model streets at the city level or street level. Because it is updated only locally in subsequent computing, the result can be improved by adapting input data and the parameters of the model. We reconstruct the entire Paris streets in a few minutes and show how the results can be edited simultaneously by several concurrent users.

Figure 45: StreetGen in a glance. Given road axes, reconstruct network, find corner arcs, compute surfaces, add lanes and markings.

INTRODUCTION

Streets are complex and serve many types of purposes, including practical (walking, shopping, etc.), social (meeting, etc.), and cultural (art, public events, etc.). Managing existing streets and planning new ones necessitates data, as planning typically occurs on an entire neighbourhood scale. These data can be fabricated manually (cadastral data, for instance, usually are). Unfortunately, doing so requires immense resources in time and people.

Indeed, a medium sized city may have hundreds of kilometers of streets. Streets are not only spatially wide, they also are very plastic and change frequently. Furthermore, street data must be precise because some of the structuring elements, like cornerstones (they separate sidewalks from roadways) are only a few centimetres in height. Curved streets are also not adapted to the Manhattan hypothesis, which states that city are organised along three dominant orthogonal directions Coughlan and Yuille, 1999.

The number and diversity of objects in streets are also particularly challenging. Because street data may be used for very different purposes (planning, public works, and transport design), it should be accessible and extensible.

Traditionally, street reconstruction solutions are more road reconstruction and are also largely oriented by the subsequent use of the reconstructed data. For instance, when the use is traffic simulation Nguyen, Desbenoit, and Daniel, 2014; Wilkie et al., 2012; Yeh, Zhong, and Yue, 2015, the focus is on reconstructing the road axis (sometime lanes), not necessarily the roadway surface. In this application, it is also essential that the reconstructed data is a network (with topological properties) because traffic simulation tools rely on it. However, the focus is clearly to reconstruct road and not streets. Streets are much more complex objects than roads, as they express the complexity of a city, and contains urban objects, places, temporary structures (like a marketplace). The precision of the reconstruction is, at best, around a metre in terms of accuracy.

Another application is road construction for the virtual worlds or driving simulations. In this case, we may simply want to create realistic looking roads. For this, it is possible to use real-life civil engineering rules, for instance using a clothoid as the main curve in highway McCrae and Singh, 2009a; Wang, Lawson, and Shen, 2014. When trying to produce a virtual world, the constructed road must blend well into its environment. For instance, the road should pass on a bridge when surrounded by water. We can also imitate real-world road-building constraints, and chose a path for the road that will minimise costs Galin et al., 2010. Roads can even be created to form a hierarchical

network Galin et al., 2011. Such generated roads are nice looking and blend well into the terrain, but they do not match reality. That is, they only exist in the virtual world.

The aim may also be to create a road network as the base layout of a city. Indeed, stemming from the seminal work of Parish and Müller, 2001, a whole family of methods first creates a road network procedurally, then creates parcels and extrudes these to create a virtual city. These methods are very powerful and expressive, but they may be difficult to control (that is, to adapt the method to get the desired result). Other works focus on control method Beneš, Wilkie, and Křivánek, 2014; Chen et al., 2008; Lipp et al., 2011. Those methods suffer from the same drawback; they are not directly adapted to model reality.

More generally, given procedural generation methods, finding the parameters so that the generated model will match the desired result is still an on-going issue (inverse procedural modelling, like in Martinovic and Van Gool, 2013 for façade, for instance).

In this work, we propose an original approach to the procedural modelling of streets : StreetGen. We start from rough GIS data (Paris road axis); thus, our modelling is based on a real road network. Then, we use a basic hypothesis and a simple road model to generate more detailed data. At this point, we generate street data for a large city (Paris); the result is one street network model. We use a widespread street network model, where the skeleton is formed by street axis and intersection forming a network. Then other constituents (lane, pedestrian crossing, markings, etc.) are linked to this network. We base all our work on a Relational DataBase Management System (RDBMS), to store inputs, results, topology, and processing methods.

In Section 3.3 we explain why we chose to base our work on a RDBMS, and explain the hypothesis, how we generate the road surface, and how the parameters of the resulting model can be edited. In Section 3.4 we provide results of street generation and results of editing. In Section 3.5, we discuss the results and present limitations and possible improvements.

METHOD

Introduction to StreetGen

The design of StreetGen is a result of a compromise between theoretical and practical considerations. StreetGen amplifies data using a simple, yet strong hypothesis. As such, the approach is to attain correct results when the hypothesis appears correct and change the method to something more robust when the hypothesis appears wrong, so as to always have a best guess result.

Second, StreetGen has been designed to work independently at different level. It can generates street data at the city level. The exact same method also generates street data interactively at the street level.

Lastly, StreetGen results are used by different applications (visualisation, traffic simulation, and spatial analysis). As such, the result is a coherent street data model with enforced constraints, and we also keep links with input data (traceability).

Figure 46 sum up StreetGen process.

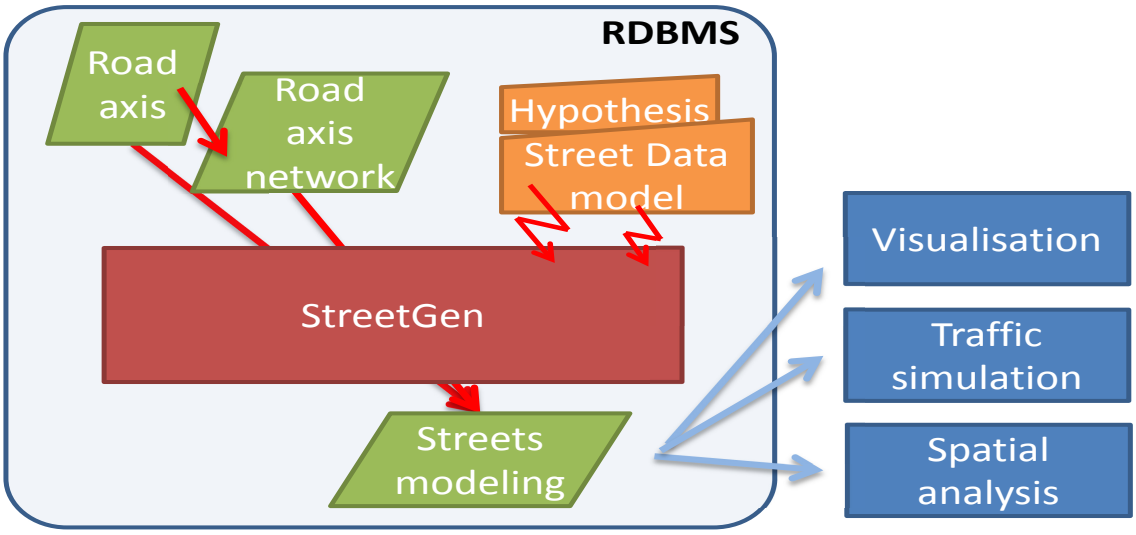


Figure 46: StreetGen workflow.

Introduction to RDBMS

We chose to use a RDBMS (*team PostgreSQL, 2014 with team PostGIS, 2014) at the heart of StreetGen for many reasons. First, RDBMSs are classical and widespread, which means that any application using our results can easily access it, whatever the Operating System (OS) or programming language. Second, RDBMSs are very versatile and, in one common framework, can regroup our input GIS data, a road network (with topology), the resulting model of streets, and even the methods to create it. Unlike file-based solutions, we put all the data in relation and enforce these relations. For instance, our model contains surfaces of streets that are associated with the corresponding street axis. If one axis is deleted, the corresponding surface is automatically deleted. We push this concept one step further, and link result tables with input tables, so that any change in input data automatically results in updating the result. Lastly, using RDBMS offers a multi OS, multi GIS (many clients possible), multi user capabilities, and has been proven to scale easily. We stress that the entirety of StreetGen is self contained into the RDBMS (input data, processing methods, and results).

StreetGen Design Principles

INPUT OF STREETGEN We use few input data, and accept that these are fuzzy and may contain errors.
The first input is a road axis network made of polylines with an estimated roadway width for each axis. We use the BDTopo[1] product for Paris in our experiment, but this kind of data is available in many countries. It can also be reconstructed from aerial images Montoya-Zegarra et al., 2014, Lidar data Poullis and You, 2010, or tracking data (GPS and/or cell phone) Ahmed et al., 2014.
Using the road axis network, we reconstruct the topology of the network up to a toler-

1 http://professionnels.ign.fr/bdtopo

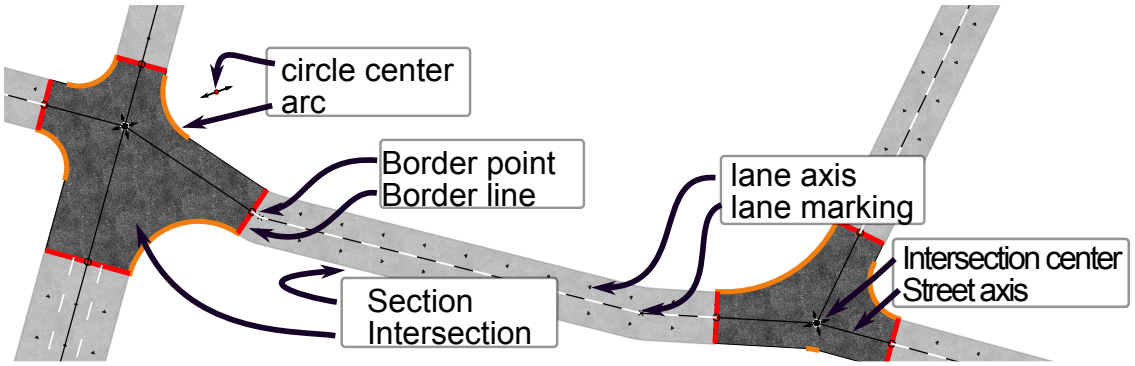


Figure 47: Street data model.

ance using either GRASS GIS (Neteler et al., 2012) or directly using PostGIS Topology team PostGIS Topology, 2014. We store and use the network with valid topology with PostGIS Topology.
The second input is the roughly estimated average speed of each axis. We can simply derive it from road importance, or from road width (when a road is wide, it is more likely that the average speed will be higher).
The third input is our modelling of streets and the hypothesis we create.

Because the data we need can be reconstructed and there is a low requirement on data quality, our method could be used almost anywhere. In particular, road attributes may be very basic and can still be corrected if necessary.

STREET DATA MODEL Real life streets are extremely complex and diverse; we do not aim at modelling all the possible streets in all their subtleties, but rather aim at modelling typical streets with a reasonable number of parameters.

First, we observe that street and urban objects are structured by the street axis. For instance, a pedestrian crossing is defined with respect to the street axis. At such, we centre our model on street axes.

Second, we observe that streets can be divided into two types: parts that are morphologically constant (same roadway width, same number of sidewalks, etc.), and transition parts (intersection, transition when the roadway width increases or decreases).
We follow this division so that our street model is made of morphologically constant parts (section) and transition parts (intersection). The separation between transition and constant parts is the section limit and is expressed regarding the street axis in curvilinear abscissa.

Third, classical streets are adapted to traffic, which means that a typical vehicle can safely drive along the street at a given speed. This means that cornerstone in an intersection does not form sharp right turns that would be dangerous for vehicle tires. The most widespread cornerstone path in this case seems to be the arc of a circle, as it is the easiest form to build during public work. Therefore, we consider cornerstone path to be either a segment or the arc of a circle. This choice is similar to Wilkie et al., 2012 and is well adapted to the city, but not so well adapted to peri-urban roads, where the curve of choice is usually the clothoid (like in McCrae and Singh, 2009b), because it is actually the curve used to build highways and fast roads.

The surface of intersection is then defined by the farthest points on each axis where the border curve starts. In this base model, we add lanes, markings, etc.

Figure 48: 3 different radius size (3m, 4.9m, 7.6 m) for streets of various importancy, from real Paris data

KINEMATIC RULE OF THUMB We propose basic hypotheses to attempt to estimate the radius of the corner in the intersection. We emphasise that these are rules of thumb that give a reasonable best guess result, and does not mean that the streets were actually made following these rules (which is false for Paris for instance).

Our first hypothesis is that streets were adapted so that vehicles can drive conveniently at a given speed s that depends on the street type. For instance, vehicles tend to drive more slowly on narrow residential streets than on city fast lanes.
Our second hypothesis is that given a speed, a vehicle is limited in the turns it can make. Considering that the vehicle follows an arc of circle trajectory, a radius that is too small would produce a dangerous acceleration and would be uncomfortable. Therefore we are able to find the radius r associated with a driving speed s through an empirical function $f(s) -> r$. This function is based on real observations of the French organisation SETRA (SETRA, 2006) (For function, see Section 1).

From our street data model and these kinematic rules of thumb, we deduce that if we roughly know the type of road, we may be able to roughly estimate the speed of the vehicles on it. From the speed, we can estimate a turning radius, which leads to the roadway geometry (See Figure 48).

Schematically, we consider that a road border is defined by a vehicle driving along it at a given speed, while making comfortable turns.

GOAL The hypotheses in the above section allow us to guess a turning radius from the road type. This turning radius is used to reconstruct the arcs of a circle that limits the junctions. The method must be robust because our hypotheses are just best guesses and are sometime completely wrong.

Given two road axis (a_1, a_2) that are each polylines, and *not* segments), having each an approximate width (w_1, w_2) and an approximate turning radius ($r = \min(r_1, r_2)$, or another choosing rule), we want to find the centre of the arc of the circle that a driving vehicle would follow.

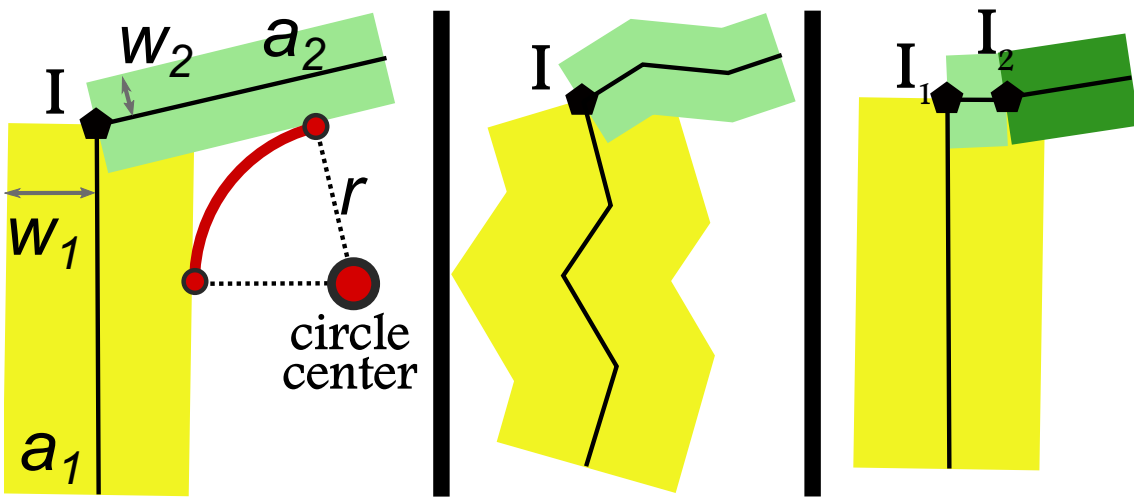


Figure 49: Finding the circle centre problem. Left classical problem, middle and right using real-world data.

METHOD Our first method was based on explicit computing, as in Wang, Lawson, and Shen, 2014, Figure 13. However, this method is not robust, and has special cases (flat angle, zero degree angle, one road entirely contained in another), is intricately two-dimensional (2D), and, most importantly, cannot be used on poly-lines. Yet real-world data is precisely made of poly-lines, due to data specification or errors.

We choose to use morphological and boolean operations to overcome these limitations. Our main operators are positive and negative buffers (formally, the Minkowski sum of the input with a disk of given size) as well as the surface intersection, union, etc.

We are looking for the centre of the arc of the circle. Thus, by definition the centre could be all the places of distance of $d_1 = w_1 + r$ from a_1 and distance of $d_2 = w_2 + r$ from a_2.

We translate this into geometrical operations:

- $\mathtt{buffer}_i$, buffer of a_i with d_i
- $\mathtt{inter}$, the intersection of boundary of buffers, which is commonly a set of point but can also be a set of points and curve. All those place could be circle centre.

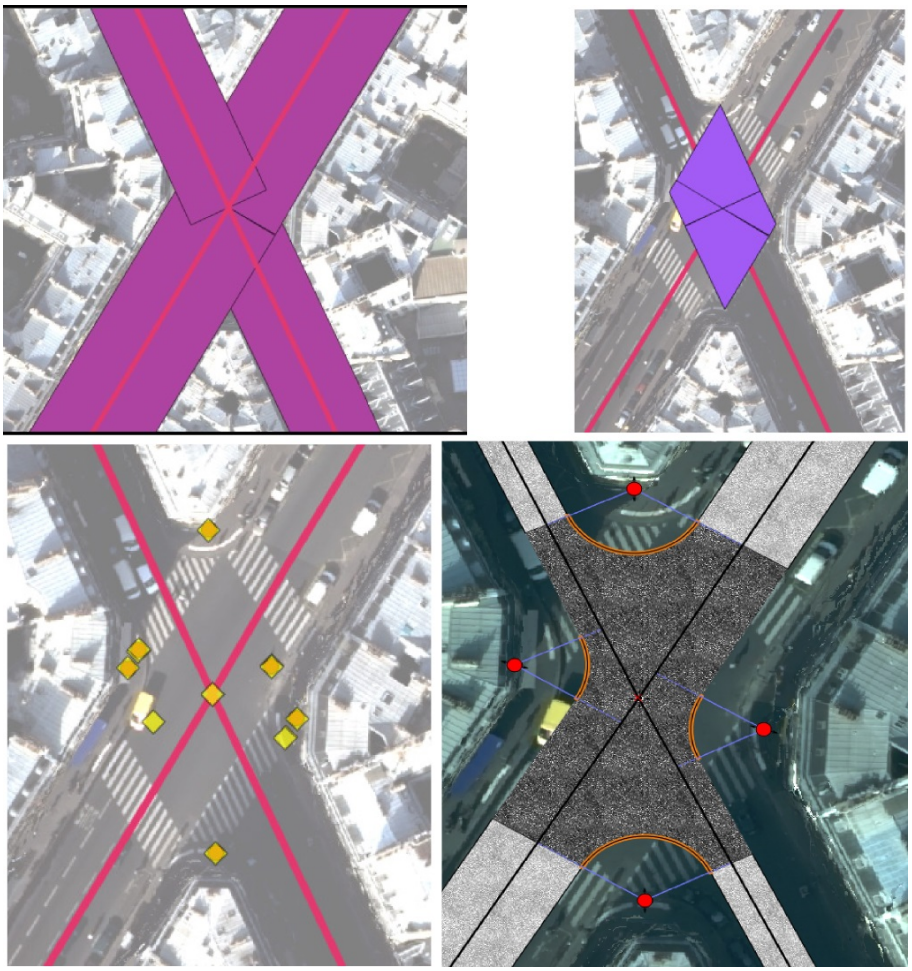


Figure 50: A method to robustly find circle centres using geometric operations. Buffer of axis is computed, then the intersection of outer ring of buffer returns a set of candidates points. Among the candidates, the one closest to the intersection centre are the final circle centre.

- `closest`, the point of `inter` that is the closest to the junction centre. We must filter this among the candidates in order to keep only the one that makes the most sense, given our hypotheses.

WHEN HYPOTHESIS ARE WRONG In some cases `closest` may be empty (when one road is geometrically contained in another considering their width for instance). In this case our method fails with no damages, as no arc is created.
The radius may not be adapted to the local road network topology. This predominantly happens when the road axis is too short with respect to the proposed radius. In this case, we reduce the guessed radius to its maximal possible value by explicitly computing the maximum radius if possible.
It also happens that the hypotheses regarding the radius are wrong, which creates obviously misplaced arcs. We chose a very simple option to estimate whether an arc is misplaced or not and simply use a threshold on the distance between the arc and the

centre of the intersection. In this case, we set the radius to a minimum that corresponds to the Paris lane separator stone radius (0.15 m).

Computing Surfaces from Arc Centres

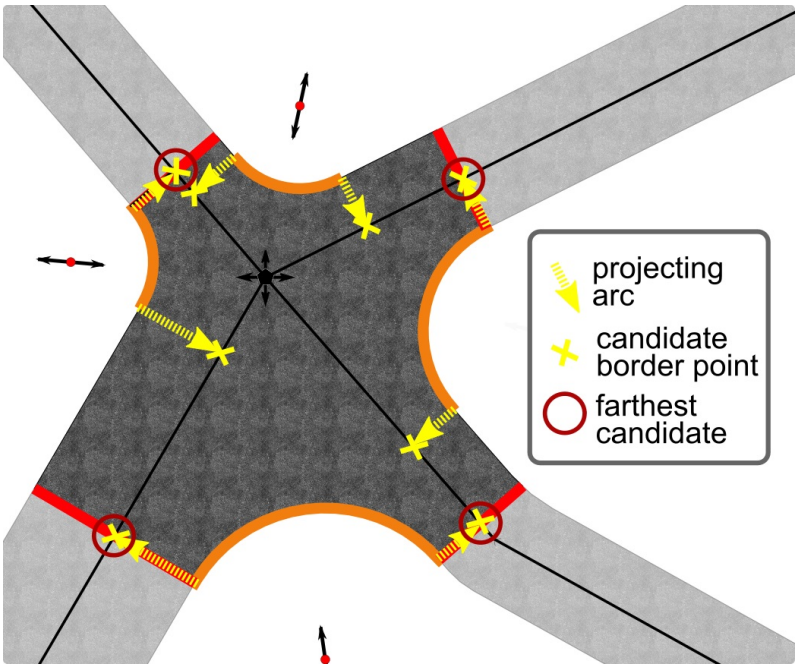


Figure 51: From circle centres, border points that limit the intersection are found by projection and filtering (farthest from intersection centre).

BORDER POINTS When centre of circles are found, we can compute the associated arcs and find intersection limit (See Fig. 51). We create the corresponding arc of circles by projecting the centres of the circle on both axis buffered by w_i. In fact, we do not use a projection, as a projection on a polyline may be ill-defined (for instance projecting on the closest segment may not work). Instead, we take the closest point.

Similarly, we 'project' the circle centre onto the road axis. We call these projections candidate border points. We have two or less border points per axis per intersection. According to our intersection surface model, we only keep one of the candidates per axis per intersection, choosing the candidate that is the farthest from the intersection centre. We define the distance from the intersection centre by using the curvilinear abscissa, which is necessary because, in some odd cases, the Euclidian distance may be misleading.

SECTION AND INTERSECTION SURFACE We compute the section surface by first creating border lines at the end of each section out of border points. The border lines are normal to a local straight approximation of the road axis. Then, we use these lines to cut the bufferised road axis to obtain the surfaces of road axis that are within the intersection.

At this point, it would be possible to construct the intersection surface by linking border lines to arcs, passing by the buffered road axis when necessary. We found it

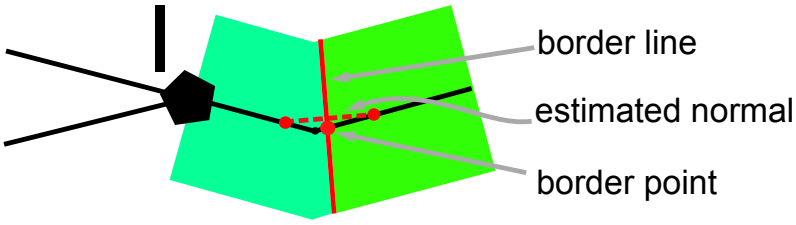


Figure 52: Creating the border line by cutting the section following a local estimation of the normal.

too difficult to do it robustly because some of the previous results may be missing or slightly false due to bad input data, wrong hypotheses or a computing precision issue.

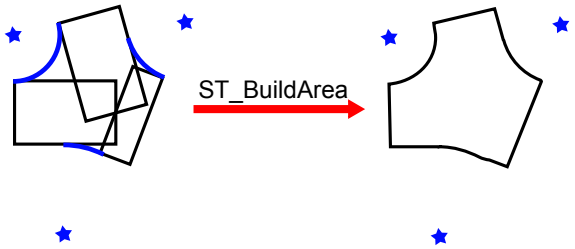


Figure 53: From cutted axis surfaces and arcs, the function ST_BuildArea build the maximum area possible.

We prefer a less specific method. We use the "ST_BuildArea" function (See Fig. 53). Given a set of geometries, it breaks all the geometries into polylines, then creates largest possible surface from those polylines. We use it on cut roads and arcs.

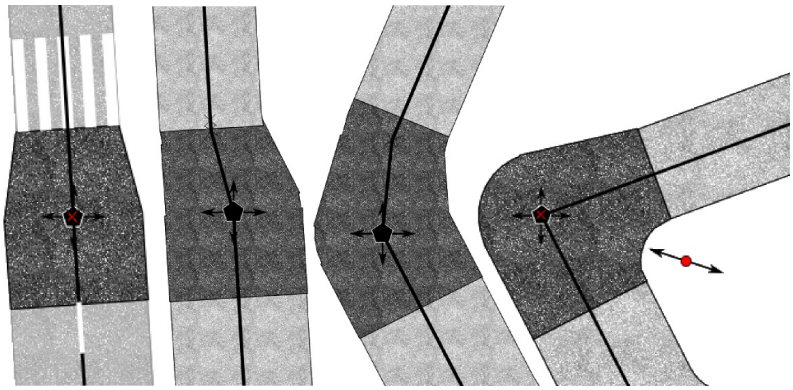


Figure 54: Variable buffer for robust roadway width transition.

VARIABLE BUFFER In the special case where the intersection is only a change of roadway width, the arc of the circle transition is less realistic than a linear transition. We use a variable buffer to do this robustly. It also offers the advantage to being able to control the three most classical transitions (symmetric, left, and right) and the transition length using only the street axis.

We define the variable buffer as a buffer whose radius is defined at each vertex (i.e., points for linestring). The radius varies linearly between vertices. One easy, but inefficient solution to compute it is to build circles and isosceles trapezoids and then union the surface of these primitives. We use the easy version.

LANE, MARKINGS, STREET OBJECTS Based on the street section, we can build lanes and lane separation markings. To this end, we cannot simply translate the centre axis because axis are polylines (See Fig. 55). Instead, a function similar to a buffer has to be used ("ST_OffsetCurve").

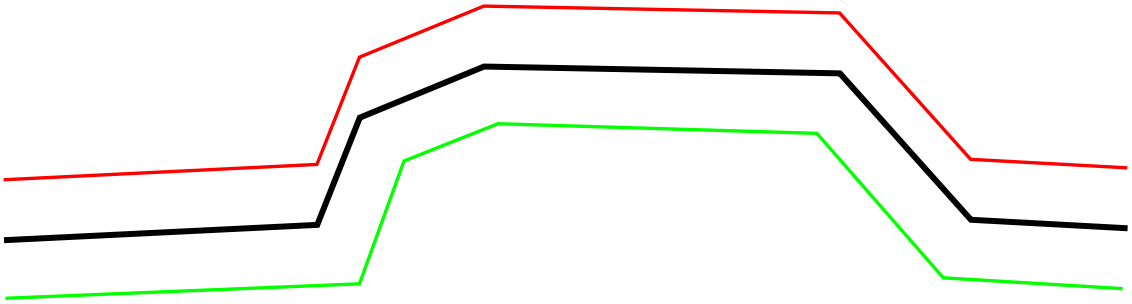


Figure 55: Starting from center line (black), a translation would not create correct a lane (red). We must use the buffer (green).

Our input data contains an estimation of the lane number. Even when such data is missing, it can still be guessed from road width, road average speed, etc., using heuristics. The number of lane could also be retrieved from various remote sensing data. For instance, Jin, Feng, and Li, 2009 propose to use aerial images. We can also build pedestrian crossings along the border lines.

Using intersection surfaces and road section surfaces, we build city blocks (See Fig. 56). We define crudely a city block surface as the complementary surface to its bounding road surfaces and road intersections. However, because all the road surface surrounding a city block may not have been generated, we use the road axis instead the road surface as city block limit when road surface is missing.

Because the road axis network has been stored as a topology, getting the surface formed by the road axis surrounding the desired block is immediate. Then, we use Boolean operations to subtract the street and intersection surfaces from the face. This has the advantage that this still provides results when some of the street limiting the block have not been computed, which is often the case in practice. By definition, the universal face ("outside") is not used as a city block!

Concurrency and scaling

The aim of this work are to model streets for a whole city in a concurrent way (that is several process could be generating the same street at the same time). Our choice of method is strongly influenced by those factors, and we use specific design to reach those goals, which are not accessory but essential.

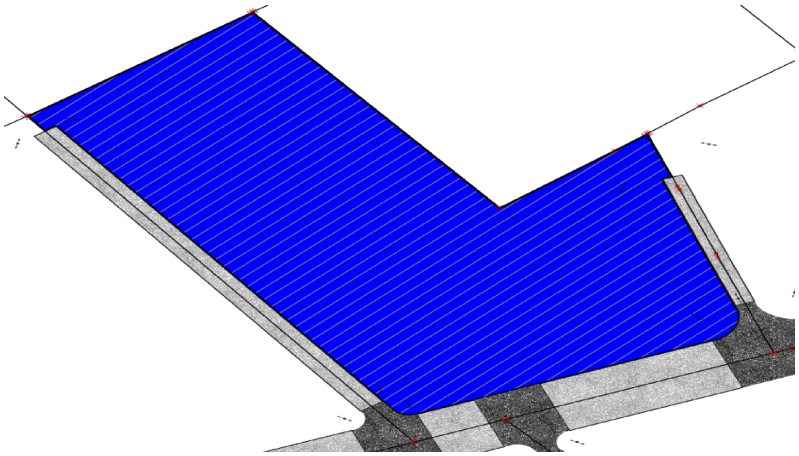


Figure 56: We generate city blocks by computing the surface that is bounded by associated road surface, road intersection, and road axis when no road surface is available (top of illustration).

ONE BIG QUERY We emphasize that StreetGen is one big SQL query (using various PL/pgSQL and Python functions).
The first advantage it offers is that it is entirely wrapped in one RDBMS transaction.This means that, if for any reason the output does not respect the constraints of the street data model, the result is rolled back (i.e., we come back to a state as if the transaction never happened). This offers a strong guarantee on the resulting street model as well as on the state of the input data.

Second, StreetGen uses SQL, which naturally works on sets (intrinsic SQL principle). This means that computing n road surfaces is not computing n times one road surface. This is paramount because computing one road surface actually requires using its one-neighbours in the road network graph. Thus, computing each road individually duplicates a lot of work.

Third, we benefit from the PostgreSQL advanced query planner, which collects and uses statistics concerning all the tables. This means that the same query on a small or big part of the network will not be executed the same way. The query planner optimises the execution plan to estimate the most effective one. This, along with extensive use of indexes, is the key to making StreetGen work seamlessly on different scales.

ONE COHERENT STREETS MODEL RESULTS One of the advantage of working with RDBMSs is the concurrency (the capacity for several users to work with the same data at the same time).
By default, this is true for StreetGen inputs (road network). Several users can simultaneously edit the road axis network with total guarantees on the integrity of the data.

However, we propose more, and exploit the RDBMS capacities so that StreetGen does not return a set of streets, but rather create or update the street modelling.
This means that we can use StreetGen on the entire Paris road axis network, and it will create a resulting streets modelling. Using StreetGen for the second time on only one road axis will simply update the parameters of the street model associated with this axis. Thus, we can guarantee at any time that the output street model is coherent and up to date.

Computing the street model for the first time corresponds to using the 'insert' SQL statement. When the street model has already been created, we use an 'update' SQL statement. In practice, we automatically mix those two statements so that when computing a part of the input road axis network, existing street models are automatically updated and non existing ones are automatically inserted. The short name for this kind of logic (if the result does not exist yet, then insert, else update) is 'upsert'.

This mechanism works flawlessly for one user but is subject to the race condition for several users. We illustrate this problem with this synthetic example. The global streets modelling is empty. User1 and User2 both compute the street model s_i corresponding to a road axis r_i. Now, both users upsert their results into the street table. The race condition creates an error (the same result is inserted twice).

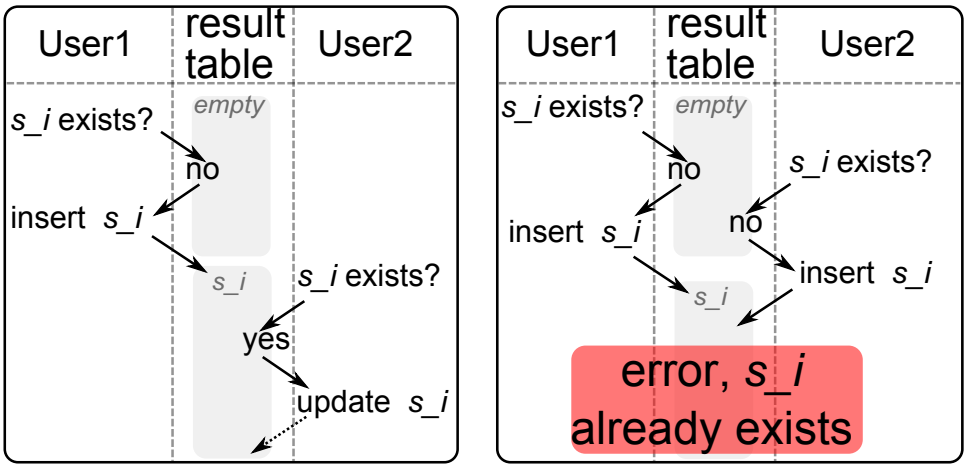


Figure 57: Left, a classical upsert. Right, race condition produces an error.

We can solve this race problem with two strategies. The first strategy is that when the upsert fails, we retry it until the upsert is successful. This strategy offers no theoretical guarantee, even if, in practice, it works well. We choose a second strategy, which is based on semaphore, and works by avoiding computing streets that are already being computed.

When using StreetGen on a set of road axes, we use semaphores to tag the road axes that are being processed. StreetGen only considers working on road axes that are not already tagged. When the computing is finished, StreetGen releases the semaphore. Thus, any other user wanting to compute the same road axis will simply do nothing as long as those streets are already being computed by another StreetGen user. This strategy offers theoretically sound guarantees, but uses a lot of memory.

Generating basic Traffic information

Introduction

StreetGen is based on tables in a RDBMS. As such, its model is extremely flexible and adaptable. We use this capacity to generate basic geometric information needed for traffic simulation. The world of traffic simulation is complex, various methods may require widely different data, depending on the method and the scale of the simulation.

For instance, a method simulating traffic nation-wide (macro simulation) would not require the same data as a method trying to simulate traffic in a city, neither as a method simulating precise trajectory of vehicle in one intersection.

Moreover, traffic simulation may require semantic data. For instance an ordinary traffic lane and the same lane reserved to bus may be geometrically identical but have a very different impact in the simulation.

Traffic simulation may require traffic light sequencing, statistics about car speed and density, visibility of objects, lighting, etc.

Therefore, we do not pretend to provide data for all kind of traffic simulations, but rather to provide basic geometric data at the scale of a city. The basic geometric information we choose to provide are lane and lane interconnection. Because lane and interconnection are integrated into StreetGen, the links between lane, interconnection and road network (road axis, intersection) is always available if necessary.

We define lane as the geometric path a vehicle could follow in a road section. A lane is strictly oriented and is to be used one-way. The intersections are trajectories a vehicle could follow while in an intersection, to go from one road section (lane) to another road section (lane). Similarly, interconnections are one-way.

Generating Lanes

Our data contains an approximate number of lane per road axis. Even in absence of such data, it could be estimated based on the road width and importance.

We compute the lanes of an axis using the buffer operation (formaly Minkowsky sum with disk), as a simple translation would not produce correct result (See Fig. 55). We create lane axis and lane separator, the second being a useful representation, and potential base to generate lane separation markings. The lane generation then depends on the parity of the number of lane, and is iterative (See Fig. 58). Special care must be taken so that all polylines generated have a coherent geometric direction.

Our data set also gives approximate information direction for each road axis. The road axis direction may be 'Direct', 'Reverse' or 'Both'. 'Direct' and 'Reverse' are both for one-way roads, with the global direction being relative to the road axis geometry direction (i.e. order of points). In 'Both' case we only know that the road is not one-way.

Please note that this simple information are very lacking to describe even moderately complex real roads (for instance, 3 lane in one direction, and one lane in the other). For lack of better solution, we have to make strong assumptions.

In the case of 'Reverse' or 'Direct', all lanes shall have the same direction. In the 'Both' case, lanes on the right of the road axis should have same direction as road axis, and lanes on the left opposite direction. In odd case, the center lane will be considered on the right of the road axis. Lane are numbered by distance to road axis, side (right first). Figure 59 gives an overview of possible lane directions. .

Generating trajectories in interconnection

Our dataset lacks any information about lane interconnection, i.e. which connexion between lanes are possible and what trajectory those connections have. For instance, being on the right lane of street X, is it possible to go to the right lane of street Y at tne next intersection, and following which trajectory?

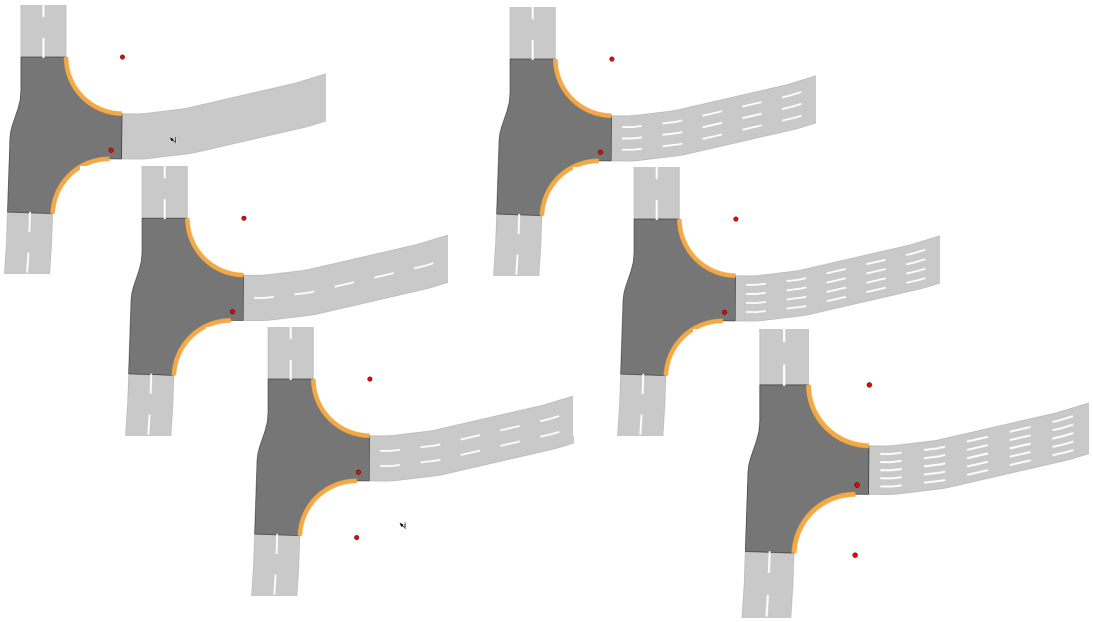


Figure 58: Generating various number of lanes, displayed in QGIS with dotted lines.

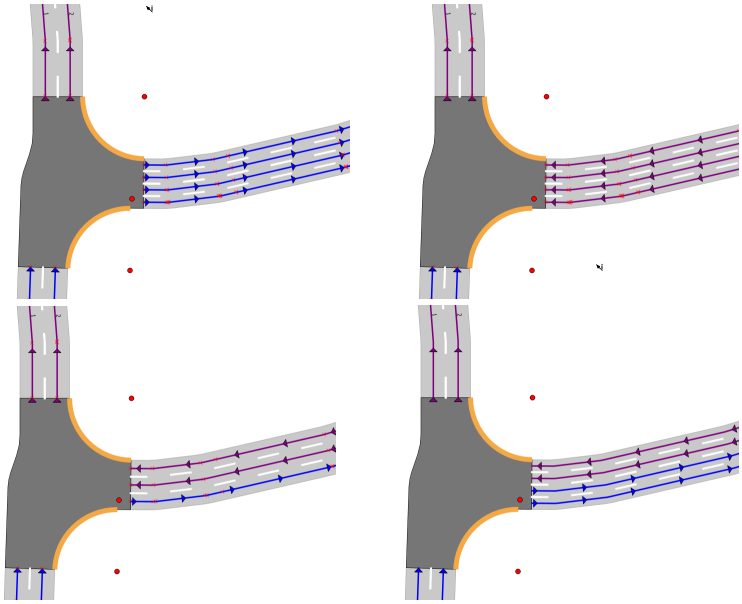


Figure 59: Default possible direction for lanes.

Strong assumptions are necessary. We use the orientation of lanes and consider that interconnection can only join lanes having opposite input direction in an intersection. Considering an intersection, each lane either comes in or out of this intersection (intersection input direction). Furthermore, we consider that lanes of the same road section are not directly connected (no turn around). Please note that in real life usage such trajectory may be possible. We create an interconnection for each pair of lanes respecting those conditions.

Actual vehicle trajectories in intersections are very complex, depending both on kinematic parameters, driver perceptual parameters, driver profile, vehicle, weather condition, etc. For instance Wolfermann, Alhajyaseen, and Nakamura, 2011 study a simple case and model only the speed profile.

We generate a plausible and simple trajectory using Bezier curves. Moreover, we isolated the part responsible for trajectory computing so it can be easily replaced by a more adapted solution than Bezier curve.

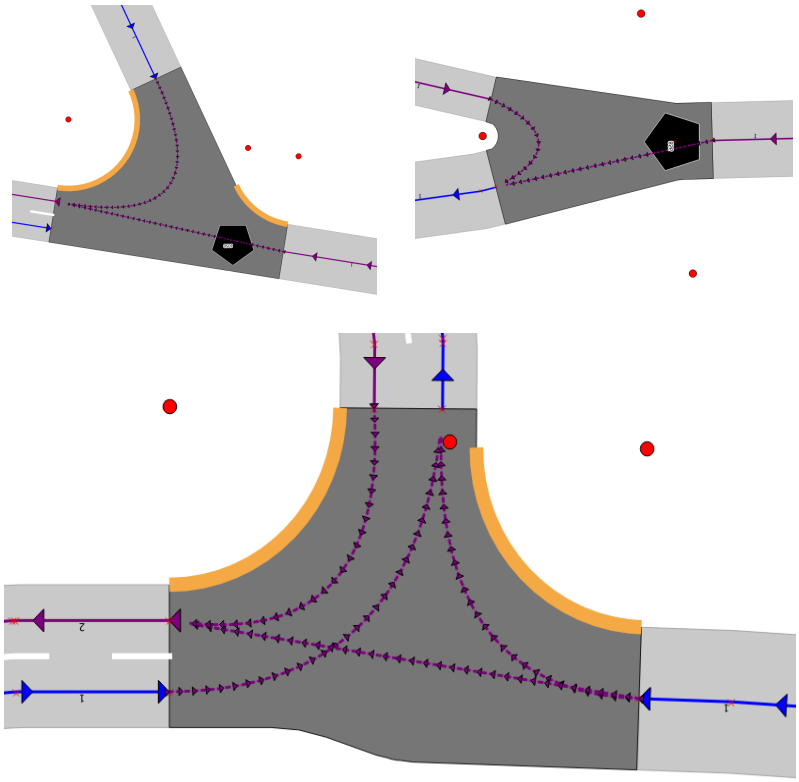


Figure 60: interconnection trajectory, Bezier curve influenced by start/end and possibly intersection centre.

Bezier control points are the points where lane center enter/exit the intersection. The third control point depends on the situation. It usually is the barycentre of lanes intersection and intersection centre. However, when lanes are parallel, lane intersection is replaced by enter/exit barycentre. In special case when lanes are parallel and opposite, the centre of the intersection is not considered to obtain a straight line trajectory. Figure 60 presents interconnection trajectory generation in various situations.

Roundabout detection

StreetGen has been used for traffic simulation. StreetGen does not consider semantic difference for any intersection. However traffic simulation tools make a strong difference between intersection and round-about.

Still, the traffic modelling is widely different between roundabout and classical intersection. Thus we need a method to detect roundabouts. We face a problem similar to (Touya, 2010, Section 3.1). The main issue is that round-about definition is based

on the driving rules in the intersection (type of priority, no traffic light,...). Yet those details are not available on the road axis network we use. If we use a strict geometric definition (round about are rounds), we could try to extract the information from aerial images (Ravanbakhsh and Fraser, 2009) or from vehicle trajectory Zinoune, Bonnifait, and Ibanez-Guzman, 2012. Yet both this example are not in street settings, where round-abouts may be much smaller, and much harder to see on aerial images. Moreover, vehicle trajectory would be much less precise because buildings mask GPS.

Of course we are far from having this level of information, therefore we used the little information available, that is geometrical shape of street axis and street names. We need a way to characterize a round-about that can be used for detection. We cannot define round-about only based on the topology of the road network (such as : a small loop), nor purely based on geometry (road axis is forming a circle) because round abouts are not necessary round. We noticed that road axis in a roundabout tends to have the same name, and/or contain the word 'PL' or 'RPT' (IGN short for 'Place' and 'Rond-Point' (roundabout)).

Therefore we use two criterias to define a potential roundabout : its road axis may be round (geometric criteria), and the road axis might have the same name or contain 'PL' or 'RPT' in their name (toponym criteria). We use Hough transform (Duda and Hart, 1972) to detect quadruplets of successive points in road axis that are a good support for an arc of circle, then perform unsupervised clustering wia DBSCAN algorithm (Pedregosa et al., 2011, Ester et al., 1996). To exploit road-name we explore the road network face by face while considering if all the road of a face have the same name and/or some contains special 'PL' or 'RPT' key words.

The final results are weighted, and are used by an user to quickly detect roundabouts (See Figure 61). .

Street Objects : From Road to Street

So far we essentially considered road modelling (road axis, road surface, intersection centre, intersection surface, lane, interconnection, etc.). However streets can be characterized by the great number and diversity of objects present in it. We use street objects in a broad sense, including street furniture as well as marking, trees, etc. Figure 62 gives examples of street objects modelling within StreetGen.

.

Generic street objects

Streets contains a great number of diverse objects. We propose to add those to StreetGen via an extensible mechanism so that the system can easily be completed with complex semantic or hierarchy in the future.

We observe that lot of street objects are related to the street axis and road surface, be it for position or orientation. For instance, a pedestrian crossing is defined relatively to the street axis and the sidewalk. Its orientation is also often related to the street axis direction (though this is not always the case).

We then design a system where objects can be linked to street axis, so as to be able to compute orientation and position accordingly. Each object position can be defined in absolute coordinates, according to street axis or according to side-walk position. If the

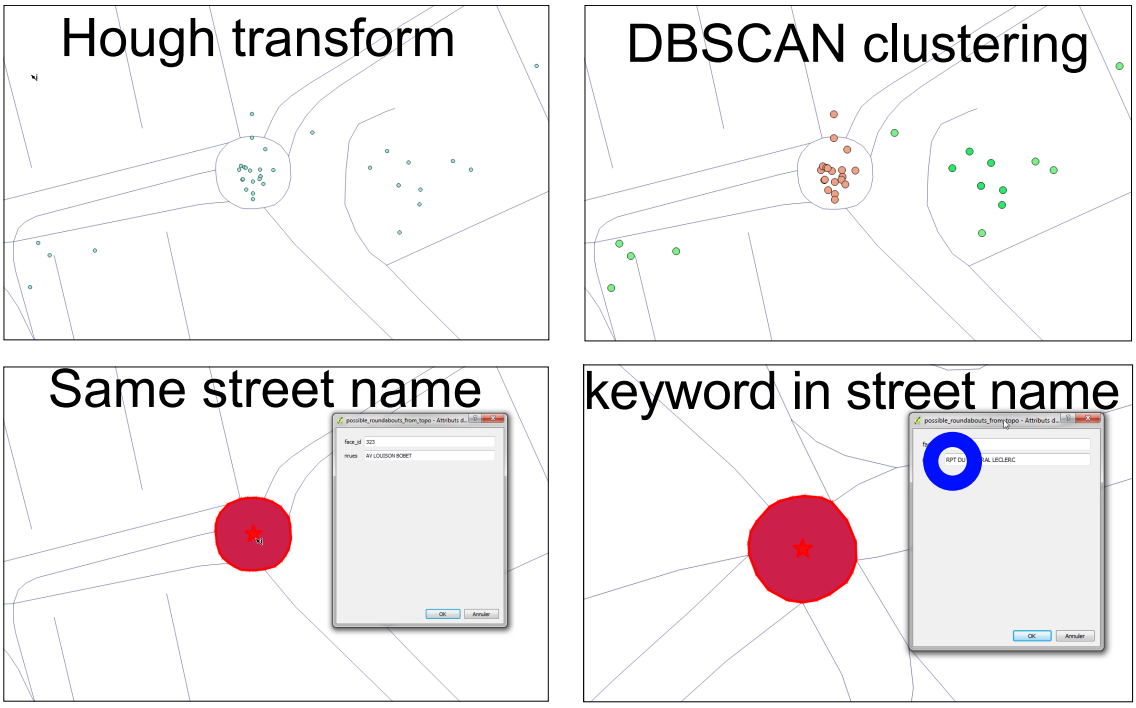


Figure 61: Roundabout detection to reduce an user work.

object is positioned relatively to street axis or sidewalk, the object position is defined by its the curvilinear abscissa along the street axis. .

Each object orientation can also be defined as absolute or according to street axis. The list of possible objects is defined in a table that can eb easily extended, and also used to build more complex information (for instance, a hierarchy). Triple store format would be good candidates for this.

Figure 63 illustrates the principle of object positioning and orientation relatively to street axis and/or street border. Lets take the example of safety barrier. In Paris those are commonly used on sidewalk, few centimeters from roadway, parallel to street axis.

In our current implementation, generic street object underlying representation is a point with semantic and possibly relative positioning and orientation information. This generic representation can be specialised for specialised street objects.

Specialised street objects

Generic street objects are numerous in streets, but many objects require more specific parameters and/or different representation than a point. We demonstrate that specialised street object can be introduced as specialisation of generic objects. The concept is borrowed from inheritance in object programming design. A new table needs to be created for each specialized street objects. This table is linked to the generic table through foreign key and add-oc triggers. The specialised table store additional parameters and additional representation, and is kept synchronised with the relevant objects of the generic table.

We demonstrate this functionality with the object pedestrian crossing (See Fig. 64).

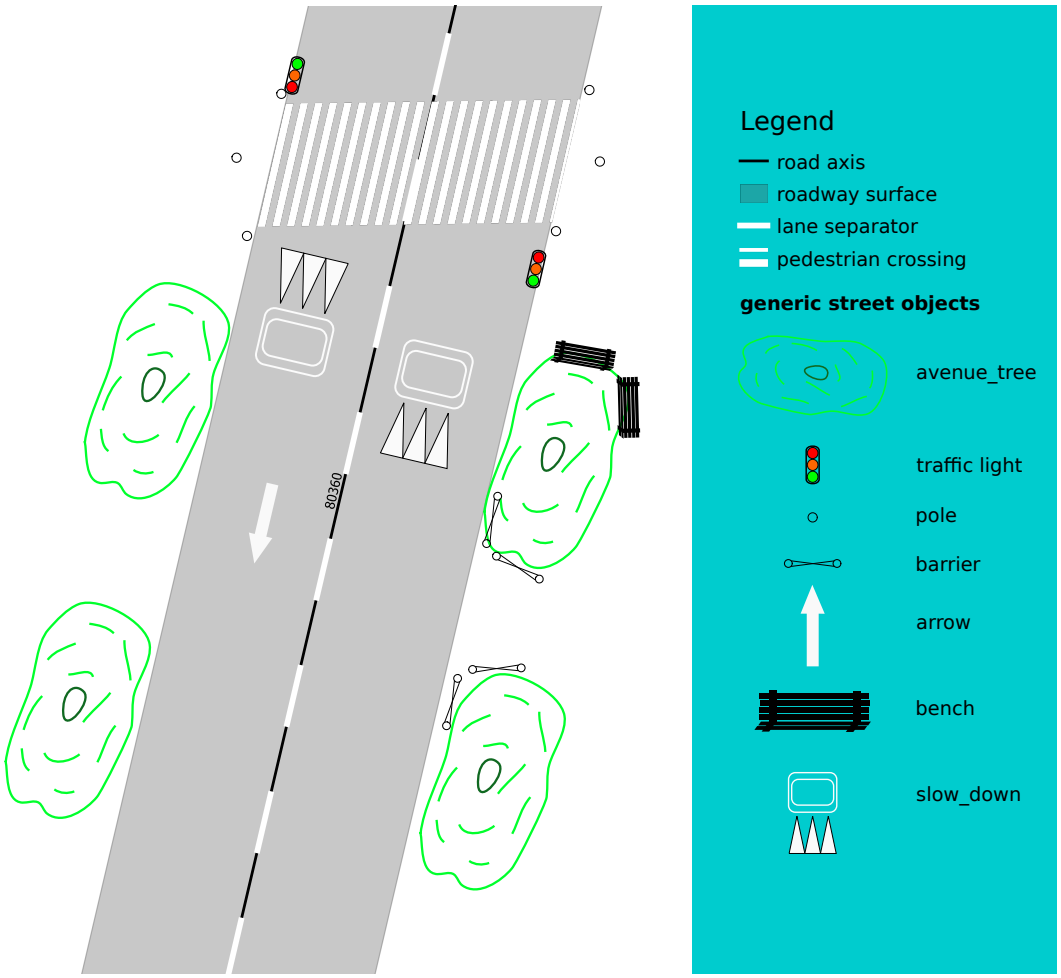


Figure 62: Street objects in StreetGen. Objects are semantic points with advanced symbology viewed in QGIS. Each object can be positioned and oriented relatively to the street axis or street border.

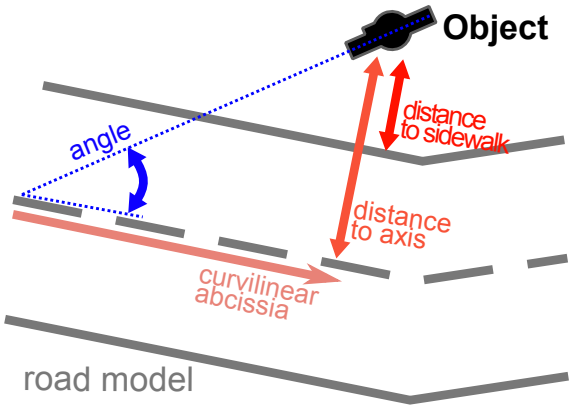


Figure 63: Object position and orientation can be defined relatively to road model.

A pedestrian crossing can be parametrized by its position along the road axis (curvilinear abscissa) and its orientation relative to the road axis. Those two parameters are al-

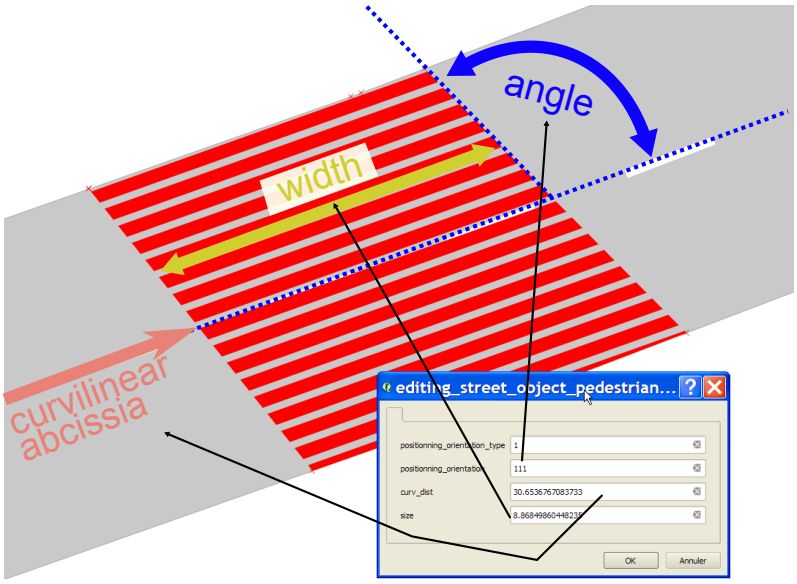


Figure 64: Specialized objects can be added, with adequate paremeters (here width) and representation (here surface displayed with qgis dash pattern).

ready defined for generic objects. However, pedestrian crossing also necessitate a width parameter, defining the width of pedestrian crossing at the road axis level. Furthermore, a pedestrian crossing is better not represented by a point and a symbol, but rather by a surface, going from sidewalk to sidewalk, which implies a special geometry (surface rather than point) and a function to generate this surface based on the pedestrian crossing parameters and the road surface.

.

Creating the pedestrian crossing surface is not immediate because road axis is a polyline. We create such function by first creating points delimitating the pedestrian crossing on the road axis. Then those points are projected left and right with an angle onto the road surface. Then the road surface border between points is extracted and sewed together to form the parallelogramoid. Figure 65 illustrates pedestrian crossing surface creation.

RESULTS

This section is dedicated to testing our street modelling method. Our road model relies on turning radius, as such we start by an experiment about estimating those turning radius. We then test the core of StreetGen, with experiments on result quality, robustness, scaling, concurrency and parallelism. The we test the traffic information generated in a real world traffic simulation application. Last, we test how generic StreetGen is by generating challenging roads, roads in another country, and an airport runway.

Estimating default turning radius

Due to lack of information about the turning radius, we had to make the assumption that turning radius depends on the type of roads (see Section 3.3.3). Although this

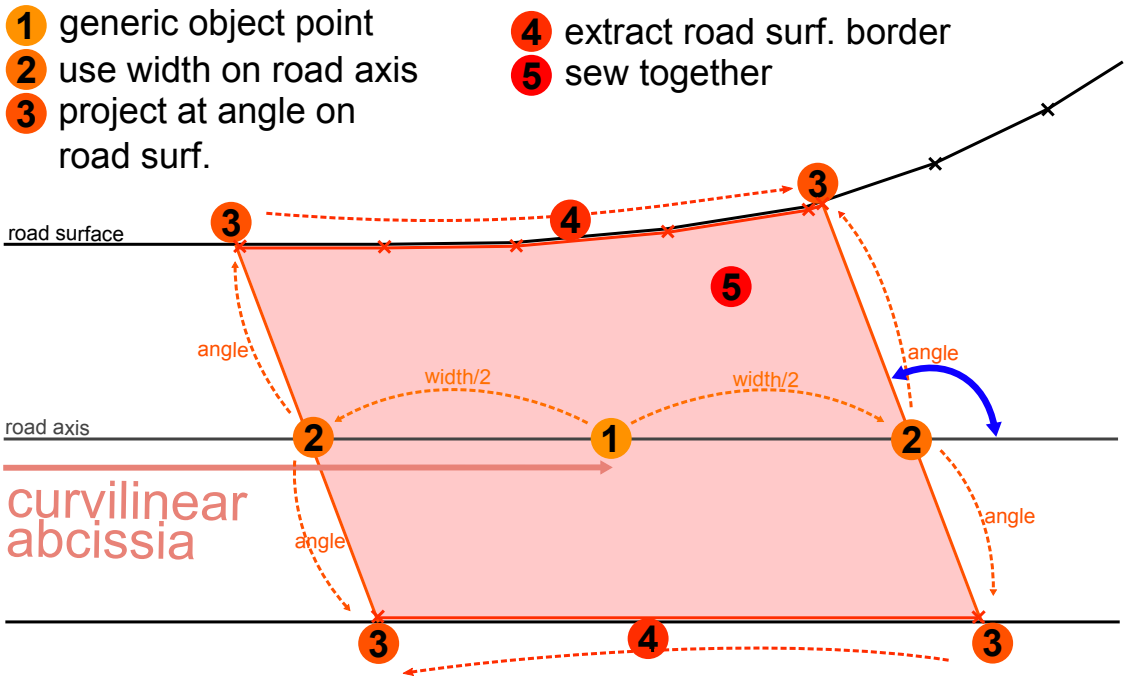


Figure 65: Building the pedestrian crossing surface based on its parameters and road axis and surface.

hypothesis is a good start and has been empirically verified outside cities, it has not been tested for city as far as we know.

Sadly radius data is not available for Paris, therefore we propose a framework to estimate this hypothesis for Paris.This test framework is illustrated in figure 66.

First we extract arc of circles from Open Data Paris 'trottoir' (sidewalk) layer using a kind of Hough transform (Duda and Hart, 1972) and filtering. Result is noisy because layer 'trottoir' contains many round objects that are not cornerstones. We obtain about 14 thousand arcs of circle. Then we map a GPS road network database containing approximate driving speed with the BDTopo road network which contains road importance, road width, number of lane, etc. We use a fuzzy geometric (how much space is shared by road axis dilated by few meters) and fuzzy semantic distance (comparing both axis street name using trigram). See Fig. 67.

Based on road data, we try several ways to predict turning radius. The first method "guesstimate" is to manually design a simple function 'radius = f_{guess}(road importance)'. The second method is to use the results of french SETRA which link average vehicle speed and turning radius for peri-urban roads, with 'radius = f_{speed}(average speed of vehicles)', using equation 1.

$$f_{speed}(speed, width) = 18.6 * \sqrt{\frac{speed}{|10.0 * width + 65.0 - speed|}} \quad (1)$$

Lastly we use machine learning to train a random forest regressor using road importance, speed, and road width to predict the radius, thus having a 'radius = $f_{rforest}$(importance, speed, road width)'. Random forest prediction is intended as a comparison to other two methods. Results are given in table 6 and illustrated in 68.

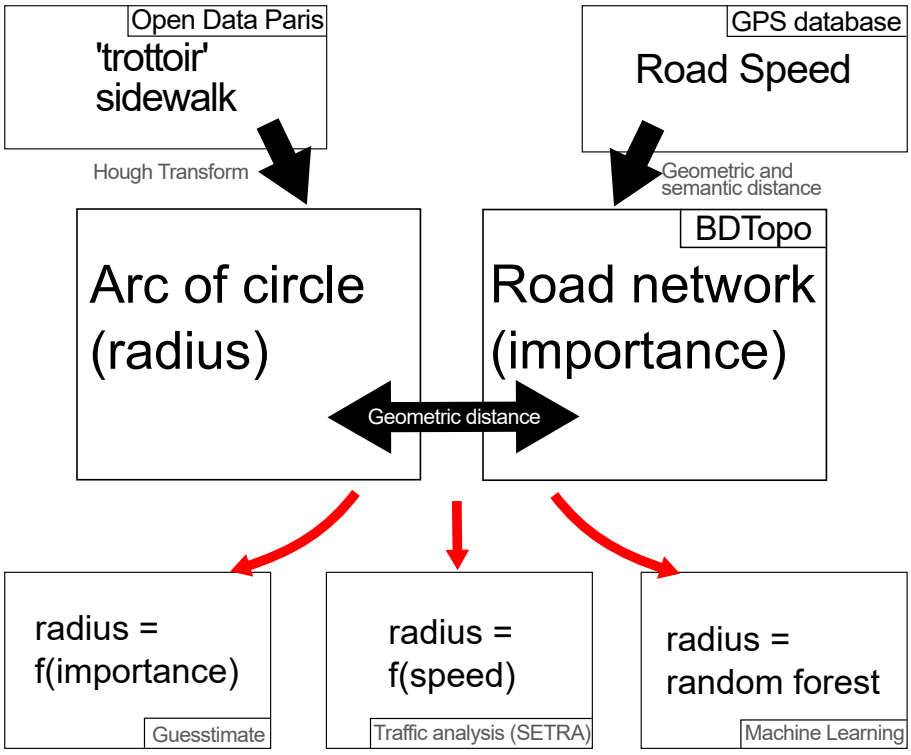


Figure 66: Workflow to test radius hypothesis.

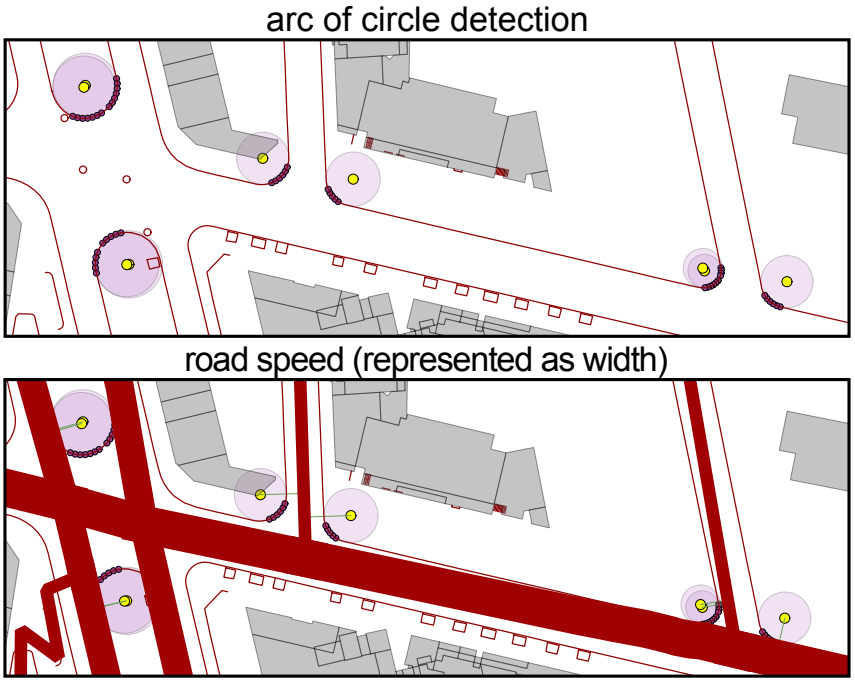


Figure 67: Radius detection and average road speed obtained by various dataset merging.

StreetGen

In this section we perform tests on StreetGen core.

Table 6: Error for various radius prediction method

metric (m)	f_{guess}	f_{speed}	$f_{rforest}$
average abs error	2.41	2.18	1.97
median abs error	1.9	1.91	1.69

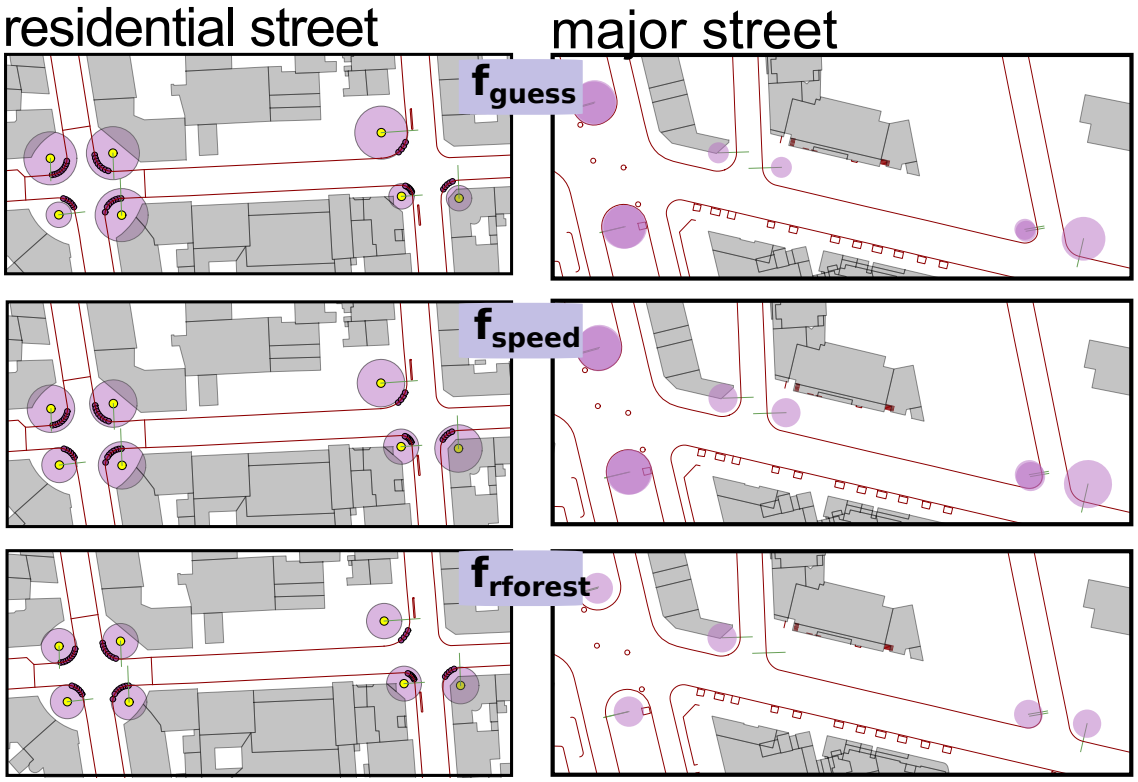


Figure 68: Illustrating radius predicted with various methods, for major and residential roads.

ROBUSTNESS Overall, StreetGen generates the entire Paris road network. We started by generating a few streets, then a few blocks, then the sixth arrondissement of Paris, then a fourth of Paris, then the entire south of Paris, then all of Paris. Each time we changed scale, we encountered new special cases and exceptions. Each time we had to robustify StreetGen. We think it is a good illustration of the complexity of some real-life streets and also of possible errors in input data.

QUALITY Overall, most of the Paris streets seem to be adapted to our street data model. StreetGen results looks primarily realistic, even in very complex intersections, or overlapping intersections.

Results are un-realistic in a few borderline cases (see Figure 70), either because of the hypotheses or the limitations of the method. Those cases are, however, easily detected and could be solved individually.

Failure 1 is caused by the fact that axis 1 and 2 form a loop. Thus, in some special cases, the whole block is considered an intersection. This is rare and easy to detect.
Failure 2 is caused by our method of computing intersection surface. In a T junction,

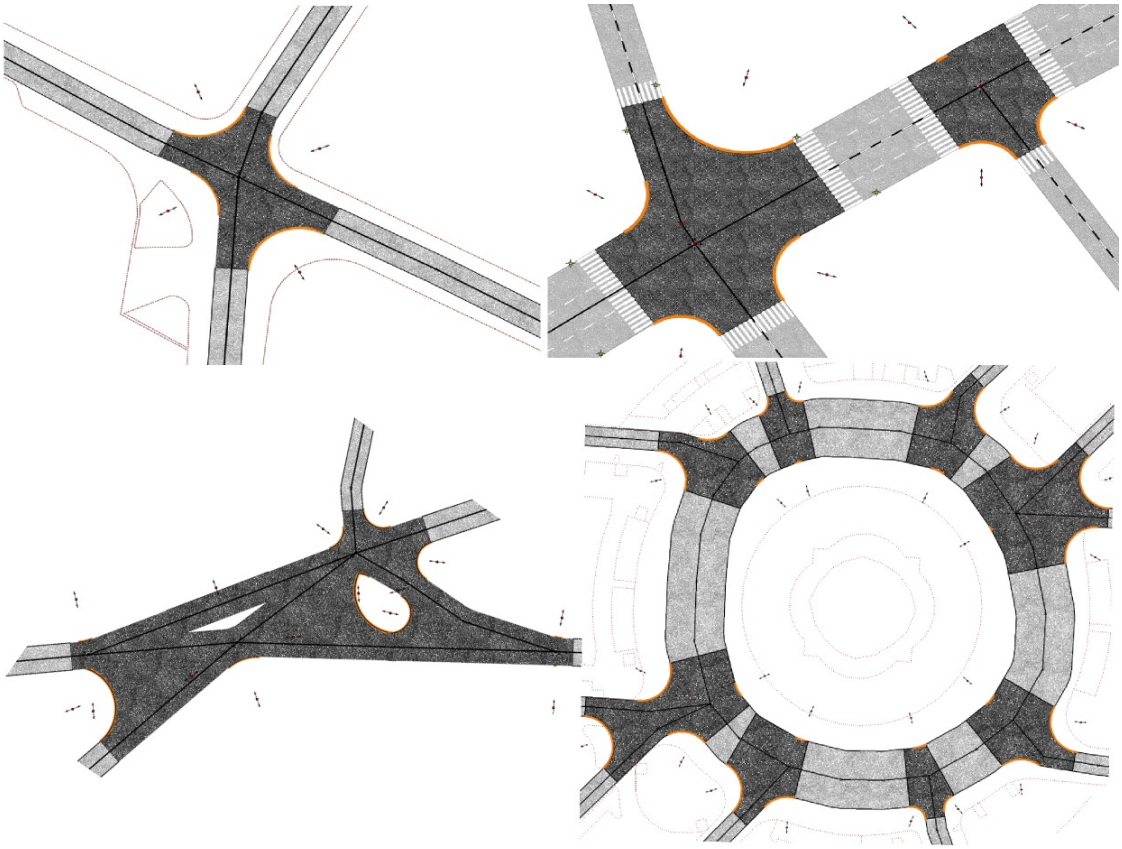


Figure 69: Example of results of increasingly complex intersection.

Figure 70: Various cases of failure from more severe to less severe (1 , 2 , 3). 1 : loop, 2 : bad buffer use, 3 : radius too big for network.

a large street orthogonal to a small street will produce a bump. It could be dealt with using the variable buffer.
Failure 3 is more subtle and happens when one axis is too short with respect to the radius. In this case, the end of the arc is way out of the intersection, because the in-

tersection is so short. It could be fixed by taking into consideration the next axis with roughly the same direction, but it would introduce special cases.

We compared the result of StreetGen with the actual roadways of Paris, which are available through Open Data Paris[2]. It clearly shows the limit of the input data, chiefly in roadway width estimations. Using interactive tools (See Fig. 71), it is possible to

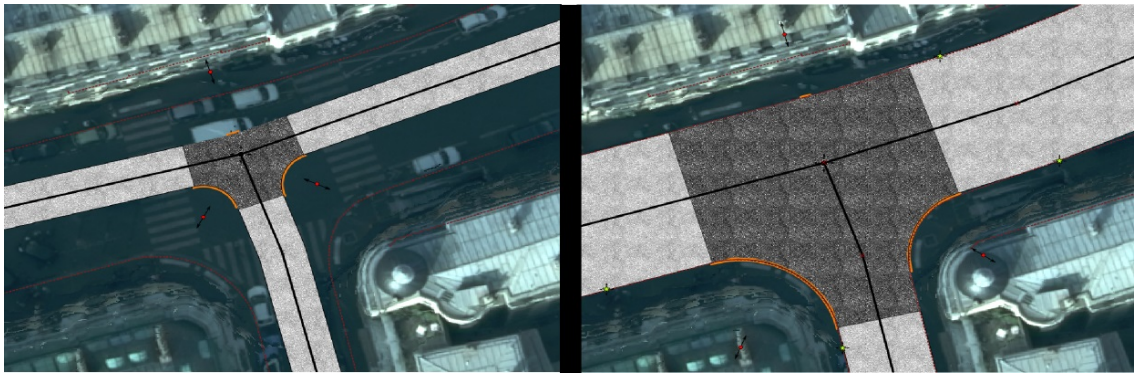


Figure 71: The estimated parameters may be far from reality (Left). It is, however, possible to manually or automatically fit the street model.

update the input data so that it is closer to the reality, until a very good match is reached. A qualitative evaluation of StreetGen result is not possible without the parameter of the road model being adapted to fit to reality, which is performed manually in Chapter 4 and automatically in Chapter 5.

SCALING The entire Paris street network is generated in less than 10 minutes (1 core). Using the exact same method, a single street (and its one-neighbour) is generated in $\sim$ 200ms, thus is lower than the human interactive limit of $\sim$ 300ms.

SQL SET OPERATIONS We illustrate the specificity of SQL (working on set) by testing two scenarios. In the first scenario (no- set), we use StreetGen on the Paris road axis one-by-one, which would take more than 2hours to complete. In the second scenario (set), we use StreetGen on all the axis at once, which takes about 10minutes.

CONCURRENCY We test StreetGen with two users simultaneously computing two road axis sets sharing between 100% and 0% of road axis. The race condition is effectively fixed, and we get the expected result.

PARALLELISM We divided the Paris road axis network into eight clusters using the K-means algorithm[3] on the road axis centroid (See Fig. 72). This happens within the database in a few seconds. Then K users use StreetGen to compute one cluster (parallelism), which reduces the overall computing time to about one minute.

2 http://opendata.paris.fr/page/home/
3 http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

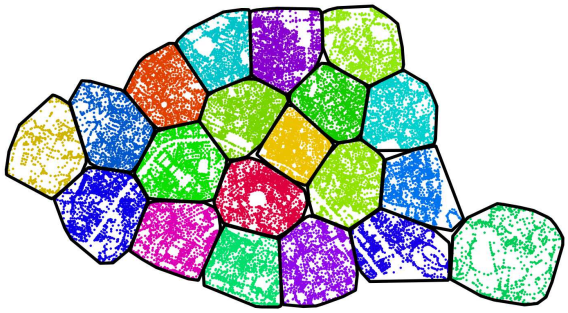


Figure 72: Clustering road axis centroid with K-Means, K=20, (black segments are convex hull).

Using Streetgen for traffic simulation

In this section we design an experiment to test the usefulness of StreetGen-generated traffic information.

Visually, generated lane and interconnection seems to be adapted in most case. The computation cost is however significantly increased because lane and interconnection are generated by triggers, and not at a global level. Moreover interconnection uses PLPython and shapely, which introduce a strong overhead.

We test StreetGen usability for traffic simulation by exporting its model for SymuVia, a traffic simulation tool (Leclercq, Laval, and Chevallier, 2007). This work was performed by Lionel Atty (IGN, SIDT) for the project TrafiPollu (Soheilian and Atty, 2016), using a mix of sql query and python modules orchestrated in a QGIS plugin.

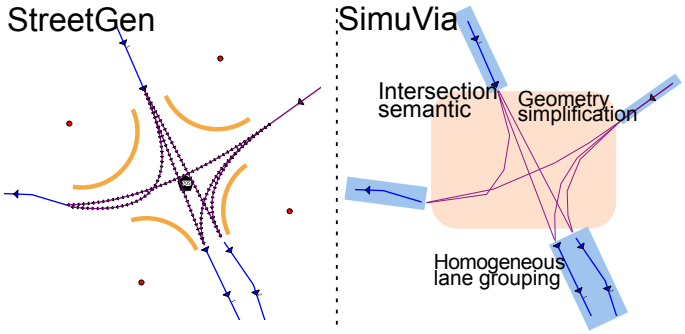


Figure 73: Converting StreetGen Traffic model to SymuVia traffic model.

The principal difficulties were different handling of intersection (SimuVia intersection model require semantic, like roundabout of classic intersection, see Section 3.3.8), necessity to regroup lanes having the same direction into homogeneous lane groups (See Fig. 73), simplification of geometries and XML export to custom SymuVia format.

Exporting is not fast (10 min for a hundred streets), but exported data is successfully used in the SymuVia traffic simulation tool.

Figure 74 proposes an example of road network traffic information manually created in a custom Symuvia tool and the same data automatically created automatically with StreetGen. Automated results where sucessfully used in SYmuvia traffic simulation tool, although it outlined the imprecison of StreetGen input data (regarding number of lanes).

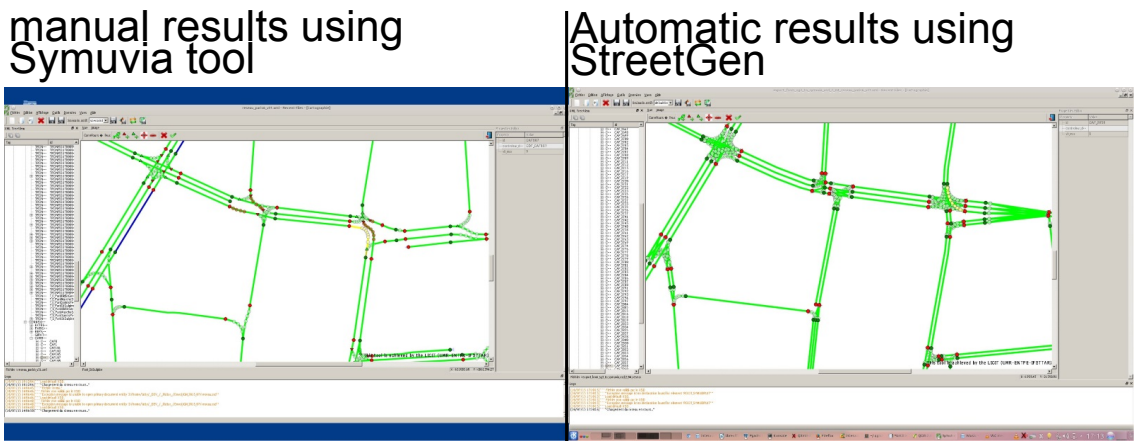


Figure 74: Manually created traffic information and StreetGen automatic traffic information.

Extending Streetgen applications

StreetGen was designed with Paris city in mind, that is many heritage roads. Indeed, street layout and characteristics vary widely around the world, and special knowledge about city type greatly helps creating adapted hypothesis. For instance a Manhattan-like grid layout is much easier to deal with.

We can still test StreetGen genericity and robustness. To this effect, we use StreetGen in different unusual scenarios.

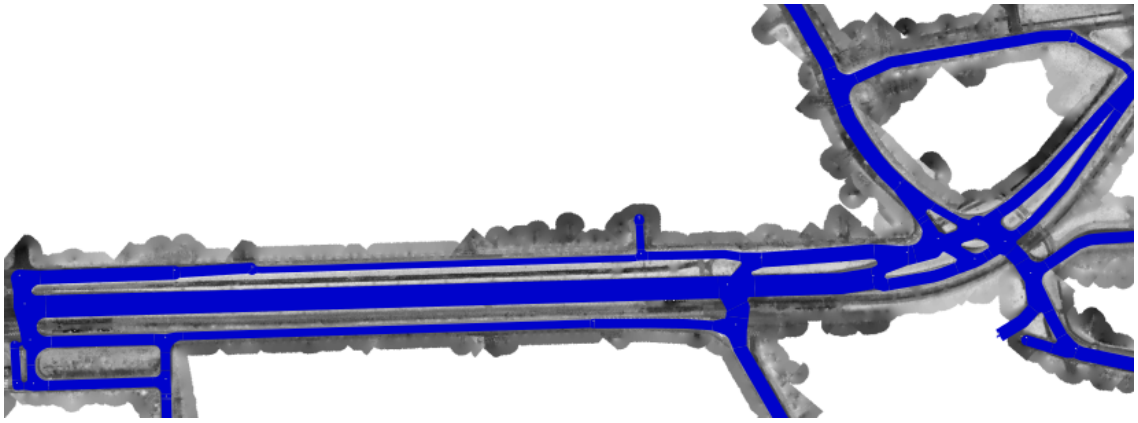


Figure 75: Experiment on StreetGen genericity : High speed intra city road, Lille, France.

INTRA CITY FAST LANE (LILLE) The first scenario is to use StreetGen to model roadway of a modern part of Lille with fast roads (See fig. 75). We stress that this example does not contains bridge or over passes, as StreetGen cannot manage those. Those roads have a modern design and as such do not necessary follow StreetGen hypothesis. StreetGen model was however sufficiently generic to model well the roadway. This road model was edited from scratch in one our, using in-base interaction (See 4).

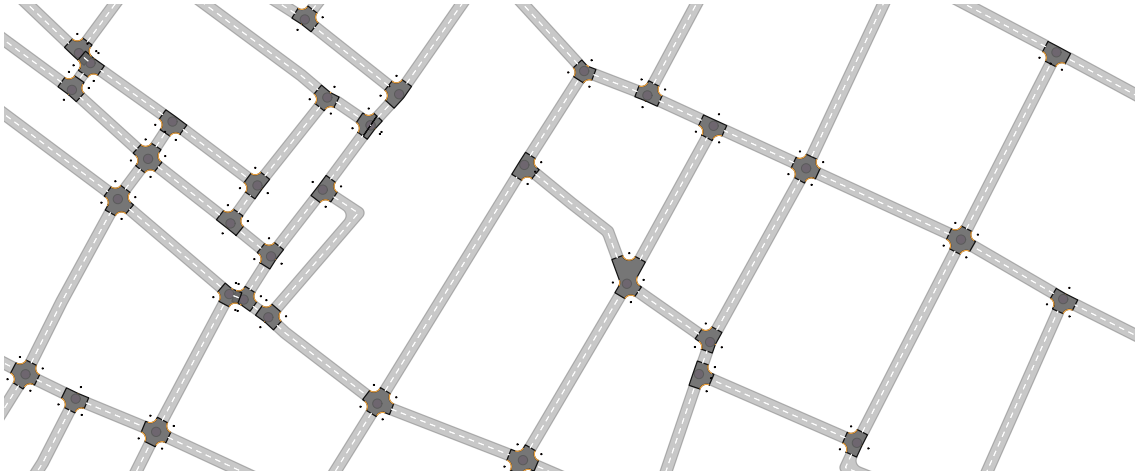


Figure 76: Experiment on StreetGen genericity : Mostly grid-based city layout, Mali

GRID-BASED LAYOUT (MALI) The second scenario is to use StreetGen for a grid-based road network (See Fig. 76). We do not have access to ground truth for this area. However, the results proved to be satisfactory for further use in 3D world building.

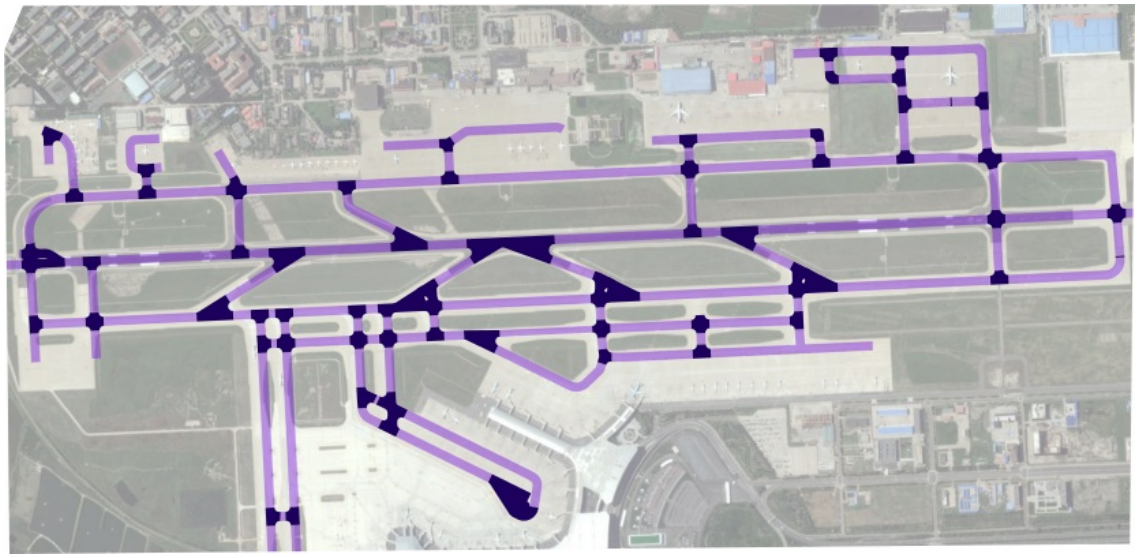


Figure 77: StreetGen used on airport runway and service roads, Bergen, Norway

AIRPORT (BERGEN) The last scenario is more stretched, and proposes to use StreetGen to generate airport runway and servicing roads (See Fig. 77). With very few exception StreetGen is able to model the runaway surface (and servicing roads) of the airport.

However because all dimensions are so different from Paris street (runway width of 40 meter is not uncommon in airport), this scenario higlighted the need to be able to change default settings easily. To cater for this need, we changed all StreetGen settings

to be stored in a global settings table. Then this glbal settings table can be adapted to each situation.

DISCUSSION

Estimating default turning radius

In Section 3.4.1 we experimented to evaluate how the turning radius could be estimated.

Results in Table 6 are pretty un-conclusive. Whatever the function to predict radius, results are poor. Radius extracted are simply too noisy, and the average speed information we extracted from road speed database lacks details (only 4 values possible). Even a random forest regressor could not work properly. Intuitively, the radius probably depends also on road construction date, historical data, neighbourhood, or other data. The SETRA function f_{speed} (1) results are however relevant when dealing with major roads (See fig. 68), but not in general.

We tested the hypothesis that maybe the function was correct but was badly parametrised. To this end, we tried to find the optimal parameters for f_{speed} using non linear least square optimisation with loss function to reduce outliers weight. We could not find better values. We take that as a proof that we do not have sufficient data to conclude about this function overall fitness for our need.

This experiment would be better performed using an adjusted StreetGen results.

Street data model

Our street data model is simple and represents the roadway well, but would need to be detailed in some aspects.
First, parking places are very abundant and important in Paris street layout, yet our model cannot specifically deal with these.
Lanes cannot have different width nor type (bus lanes, bicycle lanes, etc.).
Our model is just the first step towards modelling streets. Because we model streets, our model can not deal with bridge, tunnels, overpasses,etc. This limitations steems from the tools we use for topology management : PostGIS Topology.

Kinetic hypothesis

Overall, kinetic hypotheses provide realistic looking results, but are far from being true in an old city like Paris. Indeed, a great number of streets pre-date the invention of cars. We attempted to find a correlation between real-world corner radius (analysing OpenDataParis through Hough arc of circle detection) and the type of road or the road's average speed (from a GPS database). We could not find a clear correlation, except for fast roads. On those roads, the average speed is higher, and they have been designed for vehicles following classical engeneering rules.

Precision issue

All our geometrical operations (buffer, boolean operations, distances, etc.) rely on PostGIS (thus GEOS[4]). We then face computing precision issues, especially when dealing with arcs. Arc type is a data type that is not always supported, and thus it must be approximated by segments.

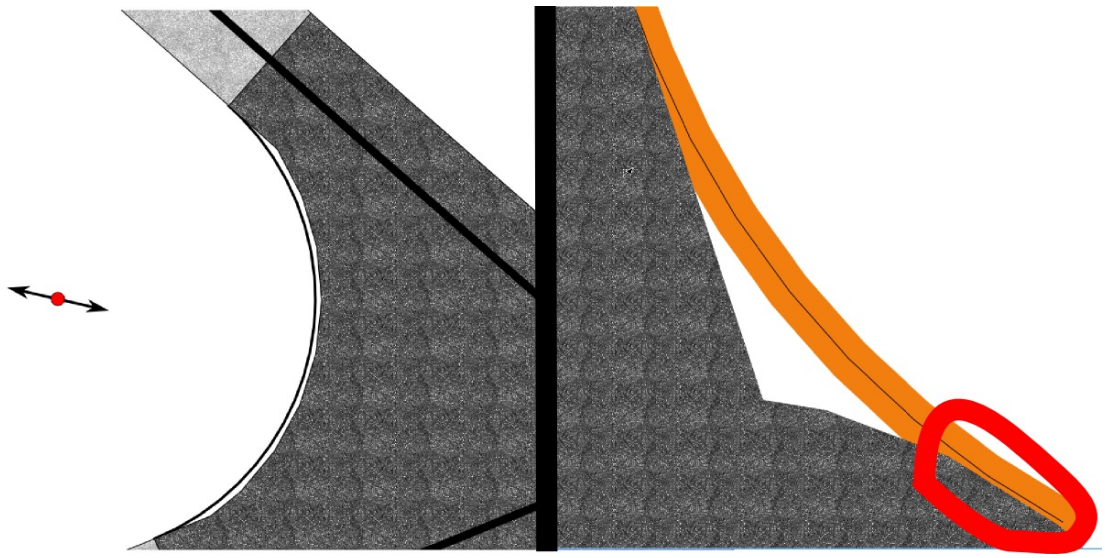


Figure 78: Example of a precision issue. Left, we approximate arcs with segments, which introduces errors. Right, the error was sufficient to incorrectly union the intersection surface.

StreetGen uses various strategies to try to work around these issues. However the only real solution would be to use an exact computation tool like CGAL The CGAL Project, 2015. It would also allow us to compute the circle centres in 3D. A recent plugin called SFCGAL[5] integrates parts of CGAL in PostGIS, using exact computing

Streetgen for traffic

Export is successful but a bit slow, although slowness is largely due to the simuvia XML format. One of the principal problem is that too many interconnections are generated. Indeed, we generate all possible interconnection, we would greatly benefit from an heuristic to generate only plausible interconnections.

Street objects

Street objects addition to StreetGen greatly improves modelling possibilities. The system was however not tested at full scale. At the city scale, storing all objects in one table may prove to be a limitation (Paris contains dozen of millions of street objects). We

4 http://trac.osgeo.org/geos/
5 www.sfcgal.org/

demonstrated point based and surface based objects, however line-based objects like markings are also very prominent.

The main limitation is however that we did not test automatic object generation based on rules and patterns. We indeed consider that correctly solving the object problem requires grammars or similar high level semantic tools which we did not try.

Extend use for StreetGen

Using a tool slightly oustide of its intended functionality is always interesting. Among the limitation, the Lille roadway possessed one road with linearly increasing road width, which can not be modelled by StreetGen. Mali dataset revealed a problem when input road axis are not properly topological. Model an airport is clearly a stretch of StreetGen capabilities. In particular, airport runways posses lots of semantic objects like lights, beacons, etc. The difference is obvious when comparing StreetGen results with an actual airport modelling (Fig. 79, courtesy of Thales TTS).

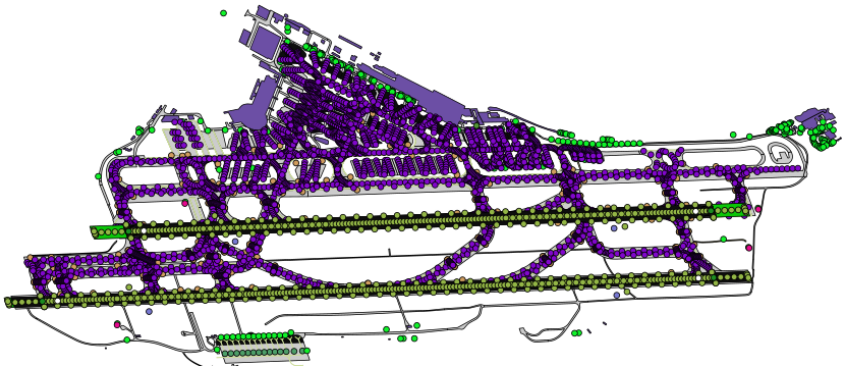


Figure 79: Real airport model, courtesy of Thales TTS

Fitting street model to reality

StreetGen was designed from the beginning to provide a best guess of streets based on very little information. However, in some cases, we want the results to better fit reality. For this, we created an interactive behaviour so that several users can fit the automatic StreetGen results to better match reality (using aerial images as ground truth for instance).

We did not created a Graphical User interface (GUI), but rather a set of automatic in-base behaviours so that editing input data or special interaction layers can interactively change the StreetGen results (See Chapter 4). Doing so ensures that any GIS software that can read and write PostGIS vector can be used as StreetGen GUI.
In some cases, we may have observations of street objects or sidewalks available, possibly automatically extracted from aerial images or Lidar, and thus imprecise and containing errors. We tested an optimisation algorithm that distorts the street model from best-guess StreetGen to better match these observations (See CHapter 5).

This subject is similar to Inverse Procedural Modeling, and we feel it offers many opportunities.

As a conclusion, we proposed a relatively simple street model based on a few hypotheses. This street data model seems to be adapted to a city as complex as Paris. We proposed various strategies to use this model robustly. We showed that the RDBMS offers interesting possibilities, in addition to storing data and facilities for concurrency.

Our method StreetGen has ample room for improvements. We could use more sophisticated methods to predict the radius, better deal with special cases, and extend the data model to better use lanes and add complex street objects managed by grammars.

In our future work, we also would like to exploit the possibility of the interaction of StreetGen to perform massive collaborative editing. Such completed street modelling could be used as ground truth for the next step, which would be an automatic method based on detections of observations like side-walks, markings, etc. Finding the optimal parameters would then involve performing Inverse Procedural Modelling.

4

INTERACTIVE CREATION AND MODIFICATION OF STREET MODEL

The goal of this thesis is to reconstruct streets, *i.e.* to create a street modelling that fits the actual streets (observations). The most immediate way to fit this model is when users directly edit the parameters of the model. To this end, a Graphical User Interface (GUI) is essential. We chose not to develop a custom GUI software because such GUI are notably difficult to design correctly, and because it strongly limits the way the street modelling can be edited. Yet the street modelling tool we use (StreetGen, Chapter 3) is entirely within a database server (PostGIS), as well as the resulting street modelling. This presents a unique opportunity, because many GIS software already exist to edit geographical data contained in a database such as PostGIS. Therefore we chose to deport interactions from the GUI software to inside the database, which allows to use any GIS software as a GUI. Additional benefits are multi-user edit capabilities.

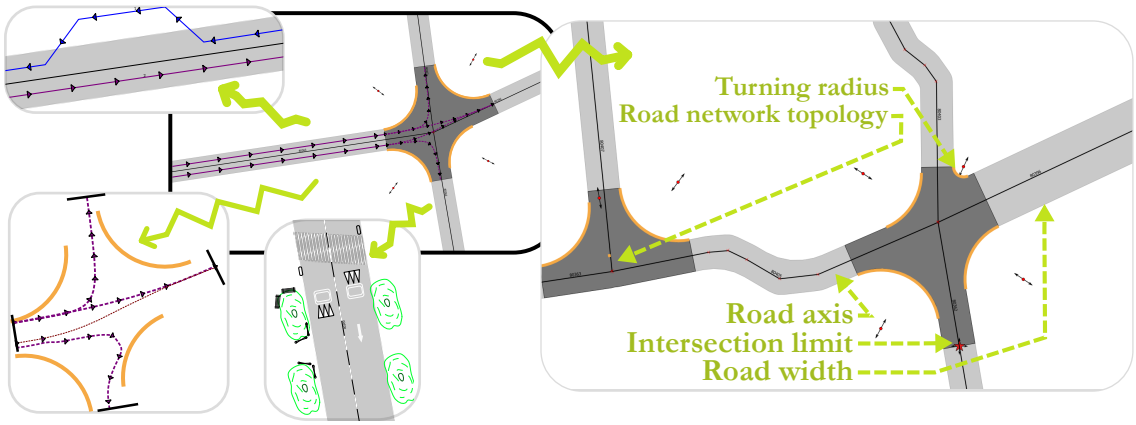


Figure 80: Interaction is handled in-base rather than in custom software. Street model is regenerated automatically (StreetGen) when user edit street model parameters using convenient and effective interactors. Road model, traffic information and street features can be edited.

Our modern world produces an increasing quantity of data, and especially geospatial data, with advance of sensing technologies, and growing complexity and organisation of vector data. Tools are needed to efficiently create and edit those vector geospatial data. Procedural generation has been a tool of choice to generate strongly organised data, yet it may be hard to control. Because those data may be involved to take consequence-full real life decisions, user interactions are required to check data and edit it. The classical process to do so would be to build an adhoc Graphical User Interface (GUI) tool adapted for the model and method being used. This task is difficult, takes a large amount of resources, and is very specific to one model, making it hard to share and re-use.

Yet many common generic GUI already exists to edit vector data, each having its specialities. We propose a change of paradigm; instead of building a specific tool for one task, we use common GIS software as GUIs, and deport the specific interactions from the software to within the database. In this paradigm, GIS software simply modify geometry and attributes of database layers, and those changes are used by the database to perform automated task.

This new paradigm has many advantages. The first one is genericity. With in-base interaction, any GIS software can be used to perform edition, whatever the software is a Desktop sofware or a web application. The second is concurrency and coherency. Because interaction is in-base, use of database features allows seamless multi-user work, and can guarantee that the data is in a coherent state. Last we propose tools to facilitate multi-user edits, both during the edit phase (each user knows what areas are edited by other users), and before and after edit (planning of edit, analyse of edited areas).

Our modern world need an increasing quantity of data, and especially geospatial data. Indeed, our capabilities to sense our environment as increased with ever more precise satellite imaging, LIDAR scanning, and mobile mapping. In parallel, another trend tends to connect data and semantize it (semantic web), with more abstract data such as vector data, becoming more accessible.

The challenge we face is then to design tools to efficiently create and edit those vector geospatial data. Generating high quality structured data is a challenge for which procedural tools are well adapted.

Procedural modelling is a powerful generative method, but notoriously hard to control (see Chen et al., 2008; Lasram, Lefebvre, and Damez, 2012; Lipp et al., 2011 for examples of increasing control). Hard control comes from the fact that understanding the link between initial parameters and the resulting model may not be obvious. Modelling is a process of simplification, as the goal is to model a complex phenomenon with a comparatively simple model.

However, having the capabilities to model something is one thing, finding the best parameters of the model so it best fits a set observation is another. The latter is called Inverse Procedural Modelling. The way to find the parameters may be a sophisticated mathematical method (Martinovic and Van Gool, 2013; Musialski and Wimmer, 2013), or a user! Moreover, whatever the level of automation, some user control is necessary, be it to validate and correct results, or to extend it beyond the limits of the procedural tools used.

Yet numerous non-procedural tools exist to edit geospatial data : GIS software. Even considering only open source software, several major GIS software exist. Unsurprisingly, each has strong points. For instance QGIS[1] has a user friendly interface and can integrate a great number of other open sources tools via plugins, GRASS GIS[2] scales very well, can be automated and has extensive raster processing, OpenJump[3] is light and has specialized tools for topology edition and validation. Leaflet[4] or Openlayer[5] allow to easily build custom light web clients to access and edit data through a browser.

Those tools have their specificities, and it would be pointless to try to create a supertool grouping all others, as modern programming paradigm tend toward simplicity (KISS principle). Users prefer to use several complementary tools to perform various tasks. However, each one of these software applications have their own programming language, User Interface (UI), and specific way to customize it. However they all have a common capability, which is to edit vector geometry and attributes.

We propose to take advantage of this common capability to use GIS software as interfaces for complex user interaction. Rather than having to create custom interaction handling for each GIS software, we deport the interaction handling inside of the database.

This approach might be coherent with recent trend to have lighter client that do not require installing (browser-based client).

1 www.qgis.org
2 https://grass.osgeo.org
3 www.openjump.org/
4 http://leafletjs.com/
5 http://openlayers.org/

This new paradigm can be used for many interactions, we use it to control an in-base Procedural Street generation method (StreetGen). As the goal is interaction, speed is important, with ideal speed under 300ms (not noticeable), with occasional spikes of a few seconds allowed.

In this work we will use both "edition" and "digitization" as the action of editing a vector layer (both geometry and attributes).

Plan

In section 4.3 we further introduce the method and the proposed in-base interaction, with details on patterns to facilitate design of in-base interaction and advanced interaction to help teamwork. In section 4.4 we illustrate how those design patterns can be used for controlling StreetGen and facilitate edits. Section 4.5 introduces perspectives and limitations, and Section 4.6 concludes this chapter.

METHOD

In this section, we start by introducing the need to interaction and control for procedural modelling methods. Then we introduce the in-base interaction concept, where the specific part of interaction handling is moved from the software to the database (Fig. 81). We present basic design patterns for in-base interaction associated with examples. Last we consider how in-base interaction could be used to help digitization, and to help plan it and analyse it afterwards.

Control of procedural modelling

Control of procedural generation tools have limited their use for a long time. Indeed, the classical workflow would be to use a procedural tool to generate a model, then manually edit the results for final details. Lets take the example of a drawing software. The goal is to generate a nice cloudy sky. Realistic clouds can be generated procedurally (using Perlin noise for instance). Once the user finds the proper parameters of the procedural clouds, he switches to fine editing, using brushes, erasers and so on to perfectly adjust clouds.

However, this approach has two major issues. The first is that manual edits are lost if the user wants to change the parameters of the procedural tool. It greatly reduces the re-usability, parameters exploration, sharing, etc.

The second issue is more modelling specific. When the user starts manual edition, the result no longer obeys the model of the procedural tool. This might no be an issue for drawing, but if the procedural tool generates a driving network, inconsiderate edition outside the procedural tool might break the topology of the driving graph or introduce errors. The obvious advantage is that by unconstraining the last human edition step, the result is not limited by the modelling space of the procedural tool.

We choose another approach where the user only makes changes through the procedural tool. We first automatically generate a modelling ('best guess'), then let the user tunes parameters of the model, as well as overrides some of the automated results. Each

time the user changes something, the relevant part of the model is re-generated at an interactive rate.

In base interaction concept

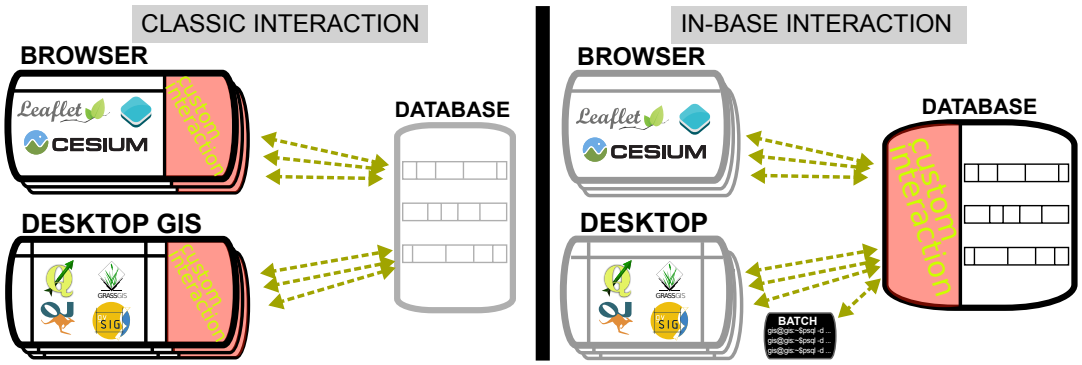


Figure 81: New proposed User Interface paradigm for GIS software. Instead of building several custom interactions for each data accessors (desktop GIS, browser GIS, etc.), we propose to use their basic vector editing (standard) and create custom interaction inside the database.

We propose a new paradigm for custom user interaction in GIS software (See Fig. 81). Traditionally, when a custom interaction is needed, GIS softwares have to be amended, often by adding a plugin, or by coding the desired interaction (web GIS). Custom interactions are therefore costly and limited to one tool. Indeed, wanting the same custom interaction for several GIS software means creating the same interaction several time so it is adapted to each GIS software. Furthermore, each custom interaction parts have to be maintained while the GIS software evolves.

For simple interaction, we propose a much simpler solution, which is to move the custom interaction handling from GIS software to database, and use the classical GIS editors (vector edition, geometry and attributes) to trigger those custom interactions. Thus, the custom interaction becomes available to any GIS software able to edit a vector in the database, thus nearly universal.

Lets take the example where a user needs a way to create grids. The classical solution would be to create a QGIS plugin (for instance) with dedicated buttons and forms to create the grid and manage it. Such a plugin would range from simple to complex, depending on how well the grids can be managed (grid fusion, etc.). The actual QGIS functionality for grids has about 15 buttons and forms. Both the UI and actual grid creation are tailored for this GIS software. On the other hand, we could automate this grid creation so that modifying a standard polygon layer produces and controls the grid (See Video 86). Then, grid creation could be performed from any GIS.

For simpler synchronising tasks, in-base interaction are even more powerful. For instance, lets consider a point layer with two orientation fields, one expressed in degrees, one in radians. Those fields have to be synchronised at all times. One solution would be to write custom handling in the GIS software, so that any change on one orientation is also done on the other. However, any changes of orientation done outside this GIS software would not synchronise orientations, thus leaving the data in an incorrect state.

Yet programming this kind of synchronisation in-base is extremely easy and efficient, it also warranties that the two orientations are always going to be seen as synchronised (ACID).

More complex in-base interactions may be needed than synchronizing two data values. Indeed, for inter-dependent values, special care must be taken to avoid useless computing and possibly circular references.

Different in-base interaction types

In-base interaction relies on triggers: functions that get executed when a table/view is modified. Thus, the mean of interaction is fixed. However, to reach scalable and safe interactions, adapted design patterns are needed. In this section we introduce those basic design ideas, which are not limited to StreetGen but are generic. In following section,those patterns are then combined to create concrete advanced interactions for a specific application (in-base street generation with StreetGen). (See Fig. 82).

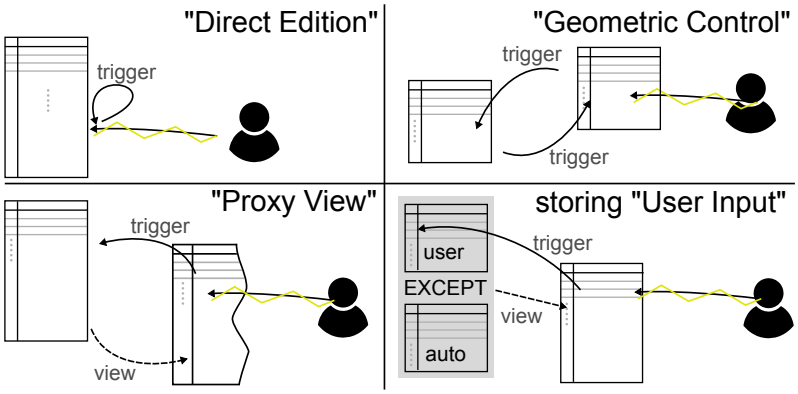


Figure 82: Various design patterns for in base user interactions. In "Direct Edition", a trigger intercepts data. In "Proxy View" a view is used as a man-in-the-middle to avoid cyclic trigger call. In "Geometric Control", another geometric object is used as a control (slider, etc.) for the targeted table. Last, storing "User Inputs" in separate table and combining it with automatically generated results solves the user input persistence problem.

"Trigger in the middle"

The simplest form of interaction is when a user directly modifies a table content. Such a modification is then processed by a trigger before the modification is applied (Trigger is between user and table, hence "trigger in the middle"). As such, it is possible to check and/or correct values modified/inserted by the user.

Lets take for example the multi-users tracking system we implemented as a QGIS plugin (See Section 4.3.4). In this system, the position and extent of the qgis user view is registered each time the user moves on the map (screen rectangle), which allows to know were the user is working, and prevent persons from working on the same area without knowing it. We observed that user editing data with QGIS never edit objects in the corner of their screen. Indeed, they tend to move the map so the object that was in the corner is approximately in the centre of the display. As such, the map seen on the

screen (rectangular) is not really the potential edit area, a rounded rectangle would be more appropriate. We create a trigger which rounds the rectangle when the rectangle is inserted into the database. See Figure 83 and web video https://www.youtube.com/v/grlkUvvSf3w?hd=1&start=120&end=134&version=3

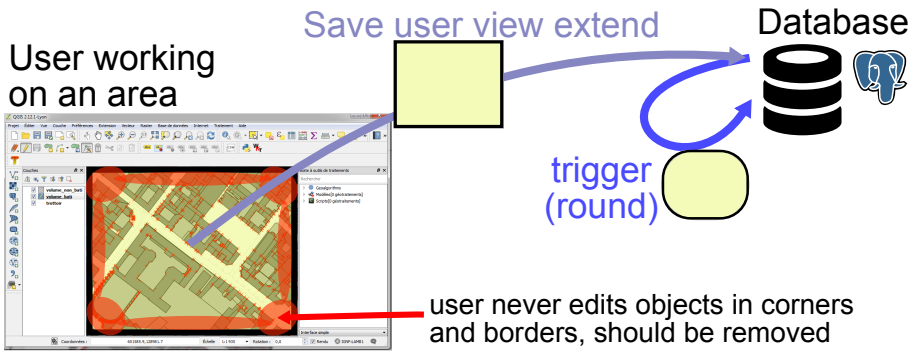


Figure 83: The position of the user map extent is recorded as a potential edit area. We notice users never edit features in the corners, which means corners are not potential edit area. We then create a trigger to round the incoming rectangle in the database, so as to have more "realistic" potential editing area.

.

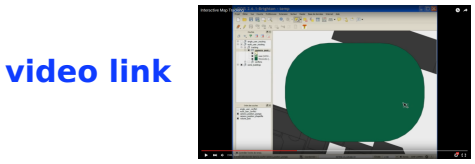


Figure 84: Video of automatic tracking of probable editing area via QGIS and PostGIS. https://www.youtube.com/v/grlkUvvSf3w?hd=1&start=120&end=134&version=3

Another common example is snapping. For instance, given a linestring representing a building footprint contour, we create a point that represent this building exit door. This point should always be on the contour. Yet when editing it, a user could move it away from the line without noticing. To prevent that, a trigger in the database first projects the edited point on the line before actually saving it.

Maybe the most common usage is for constraint enforcement. An user could modify an attribute which is constrained. For instance modifying the road width, a trigger enforce that the road width is positive (simply taking absolute value of user input) before saving it in the base. Almost all the in-base interactions we present use "Trigger in the middle".

Direct "geometric control"

"Direct Modification" imply to change one by one the object involved.

In some case, it may be much more powerful to use another geometry as a controller. The idea is fare from new, and is well adapted to database and triggers.

Lets take for instance a point cloud lens, which is defined as showing all the points within the lens geometry (See Cura, Perret, and Paparoditis, 2015b. In this case we control which points amoing billions are displayed with a geometric controller which is the lens geometry. Triggers on the lens ensure that appropriate points are displayed when change occurs. In addition, lens attributes can also be used to control other aspects. For instance a lens attribute "LOD" allows to choose which amount of points are going to be displayed. Another attribute "pass" allows to choose which vehicle pass to display (in terrestrial mobile mapping, the mapping vehicle may have made several pass at the same place at different time.)

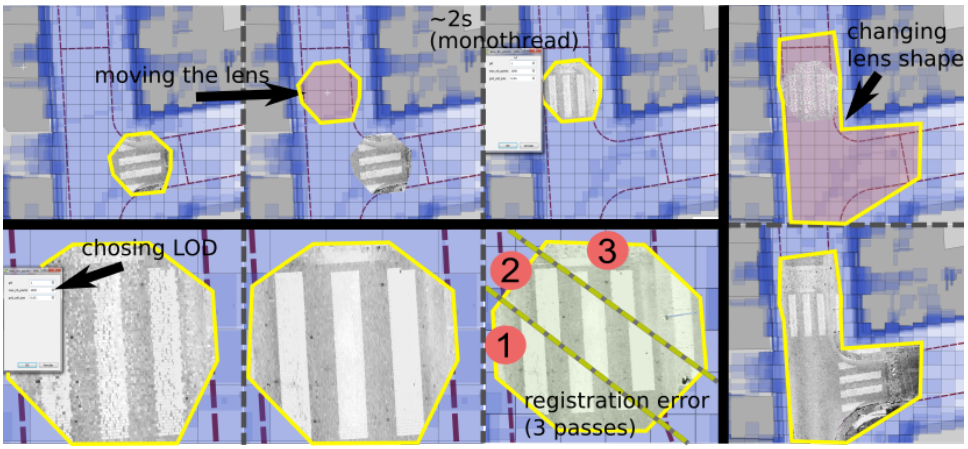


Figure 85: A GIS visualisation lens for point cloud, showed in QGIS. A lens (polygon) position and form controls what points are displayed (among several Billions points). In addition to lens geometry, lens attributes also controls other aspect of rendering, such as Level Of Detail (LOD) or the vehicle pass (temporal filtering).

Several Direct geometric control can be used conjointly, from on or several table.

Another example is the hexagonal controller discussed in 4.3.4. The goal is to generate and edit an hexagonal grid. Rather than adding hexagon by hexagon, we propose to use a direct "Geometric Control" (a polygon table with attributes) to control the hexagonal grid. The control table contains triggers, so that upon changes the hexagonal grid is accordingly created/updated. The control layer contains an attribute 'size' which control the size of the hexagons in the hexagonal grid. Such automation are easy to create and greatly simplify the control of complex objects. (See Video 86).

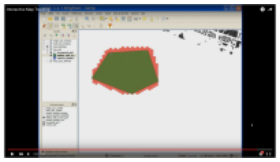
video link 

Figure 86: Using a polygon proxy to control an hexagonal grid (Database triggers), showed in QGIS..
https://www.youtube.com/v/grlkUvvSf3w&hd=1&start=288&end=323&version=3&vq=hd720

Indirect "geometric control"

Geometric control can be pushed one step further. Indeed, in both previous example, the actual geometry was directly used to control the objects (points or hexagons), that is the geometry (polygon) representing the control object was directly used. Yet, we can use geometric controller as graphical control element, that are abstracted from the map and whose geometry is not related to any geospatial meaning, like a slider.

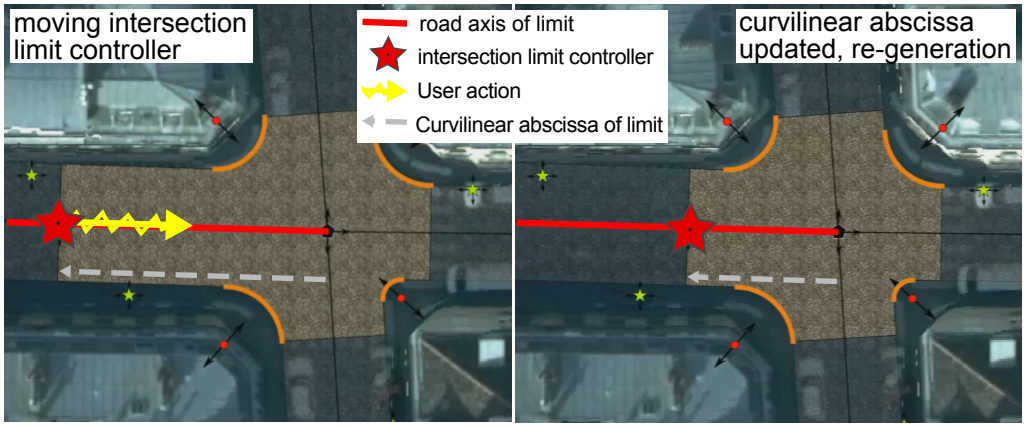


Figure 87: An indirect geometric controller permits an easy control of the position of the intersection limit which is defined by its curvilinear abscissa. Only controller changes are used to update the abscissa.

StreetGen manual intersection limit gives a good example of such a design (See Fig. 87). The goal is to allow the user to be able to choose the intersection limit, which is defined by a curvilinear abscissa along the road axis. This abscissa could be change through a form, which lacks visual feedback and is time consuming. Instead, we create an indirect geometrical controller that is a point which represents the limit. We define triggers so that a change of the controller by the user is interpreted as a change of the abscissa, which in turns triggers the regeneration of all the impacted geometries (road surface, road intersection surface, lanes, etc.). In this example, the controller is indirect as the abscissa definition is not based on the controller. More accurately, controller changes must be interpreted before having any impact.

In another example, the indirect geometric controller is used both for visualisation and control (See Figure 88). The aim is to allow the edition of Z values of a 3D polyline L_{3D} within classical 2D GIS software. Editing the Z value for each node of the polyline is currently difficult as very few software allow to directly edit it. Furthermore, in GIS software lines are drawn in the plane (seen from the top), which totally occults the Z values. We propose to edit the Z values of L_{3D} through the use of an indirect geometric controller which is the altimetry profile L_{alti} of the L_{3D} line based on L_{3D} as the origin axis. Conceptually, for each node $N_{3D}i$ in L_{3D}, we create an equivalent node N_ai in L_{alti} so that N_ai is on the perpendicular (defined on a neighbourhood) to $N_{3D}i$ at a distance of $N_{3D}i.Z - Z_{min}$, where $Z_{min} = \min_{N_{3D}i \in L_{3D}} Z(N_{3D}i)$. The user directly visualises Z values and slope. The user can edit L_{alti} nodes, moving them closer or farther from L_{3D}. Then a trigger interprets those edits in terms of new Z values for L_{3D}, which is then updated, and triggers a recomputing of L_{alti}. This idea is based on the

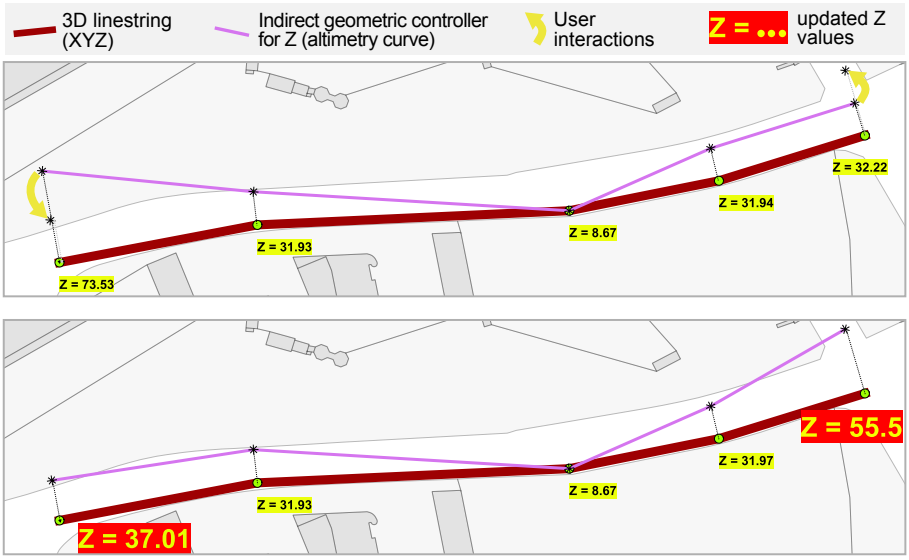


Figure 88: An indirect geometric controller is used to change the Z values of points of a 3D linestring within a traditional 2D GIS interface (here: QGIS). The altimetry curve is both a visualisation tool and an easy edit tool.

hypothesis that Z values do not vary significantly faster than X and Y values (or the altimetry curb would be very far from the initial curve.).

"Proxy view"

As seen in StreetGen manual intersection limit example (4.3.3.3), generated geometry can make a very useful indirect geometric controller. However, this introduces another issue.

The controller has a trigger that launches when the controller is edited. Yet, this edition is just a mean for the user to edit the actual value (manual intersection limit curvilinear abscissa in this case). Now editing this actual value will produce in turn the re-generation of the controller, so as to have it in a coherent state with the actual value. Yet, this re-generation of the controller is a change of its geometry, which in turn launches the trigger, etc. Thus, the risk is to enter an infinite loop of self calling triggers. The problem may be less direct, coming in the form of cyclic trigger dependencies : Trigger A launches Trigger B which launches trigger C which launches trigger A, etc.

In the controller case, the problem boils down to be able to differentiate between a change of the controller by a user, which must be interpreted and "translated", and an automated change (generation), which is not interpreted.

We propose two specialised designs to deal with this problem.

The first design we propose is the use of a "Proxy view" (or materialised view, or table), so that the user never edits the controller directly, but rather a view of the controller. That way, we know that changes coming to the controller are only those from automated regeneration, and changes coming to the view only come from the user. An additional advantage is to clearly separate the automated generation part from the human interaction part.

We illustrate this design with an indirect geometric controller that edits an intersection turning radius through a proxy view. The curbstone arc centre is generated from a radius value and other geometries. We use it as an indirect geometric controller through a "Proxy view". User edits the arc centre using this view, which is then interpreted as a new radius (smallest distance between controller and relevant road surfaces). When the new radius is updated, another trigger re-generates arc centre and other geometries based on the new radius. This re-generation updates the arc centre. If not using "Proxy view", it could have triggered the interpretation, entering into an infinite cyclic trigger call (in fact, PostgreSQL limits the number of recursions, so it simply produces an error, and not a system crash). The "Proxy view" allows to separate automated changes and changes coming from user.

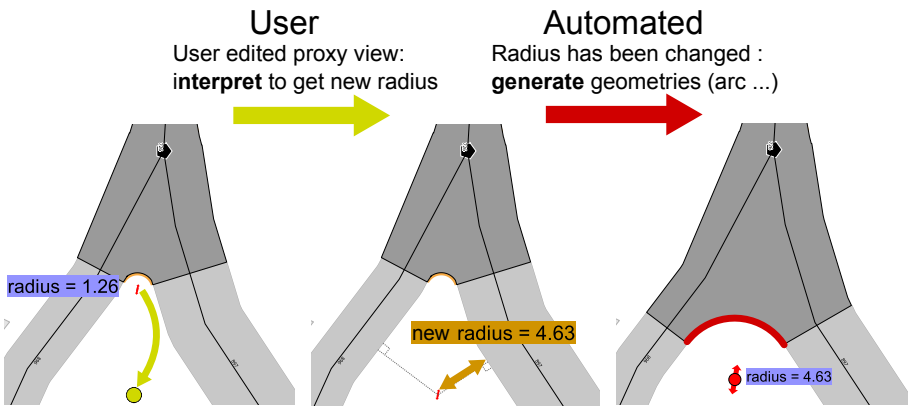


Figure 89: The curbstone arc centre is generated from the radius value and other geometries. We use it as an indirect geometric controller through a "Proxy view". The user edits the arc centre using this view, which is interpreted as a new radius. Then, another trigger re-generates arc centre and other geometries based on the new radius. No infinite cyclic trigger call occurs thanks to the proxy view which separates automated changes and user changes.

The second design to be able to separate automated changes from user interaction is much simpler. At its simplest form, it amounts to require automated changes to not only change the controller but also another column in a specific way. On the opposite, a user interaction will only change the controller, and not the other column. That way we can differentiate between the automated change and the human interaction. We use this design for indirect geometric controller for StreetGen manual intersection limit. In this case a dummy column is not needed, as the controller table also possesses the controlled value. Therefore, we simply check that both controller geometric position and curvilinear abscissa are coherent, knowing that any automated generation will synchronizes both.

Storing user choice

When mixing automated results and human input, Human interaction persistance is essential.

In StreetGen, users can modify two things : the input data (road axis, road width, etc.) , and some of the generation results (lane direction and trajectory, intersection trajectory, intersection limit, turning radius, etc.).

We consider user inputs as overrides of default values. As such, we propose to store user inputs in separate tables from automated results. This design has practical advantages. Because the user input is in a separate table, it becomes easy to save it, to merge several users inputs, etc. The user input value is stored along with a way to identify which automated result is concerned (one or several ids may be necessary).

The design is simple and can be coded in two ways using pure SQL.

The first is to use EXCEPT statement.

```
SELECT id, value
FROM user_override
EXCEPT
SELECT id, value
FROM automated_results ;
```

However such statement is not handy, as the columns of the table before and after "EXCEPT" statement must match.

Another more practical solution is to join user and automated tables, then use COALESCE, which might end up to be more costly but is more adaptable. COALESCE(value1, value2 ..) is a SQL function returning the first non null argument. Using this function allows to express the condition: if a user value exists for this object, use it, else use the automated value.

```
SELECT id,
 COALESCE( user_override.value, automated_results.value)
FROM automated_results
 LEFT OUTER JOIN user_override USING (id);
```

We illustrate user input persistance for StreetGen lane edition. By default, lanes are generated according to the street axis direction and their position regarding the road. Lane geometry is obtained by offsetting the road axis curve. Users can override the lane direction. In this case a new row is inserted into the user input lane table. This row stores the new value chosen by the user (the lane should be in the other direction). If the road axis is edited, a new lane geometry will still be automatically generated, as the user only override direction and not geometry. If the user also edits the lane geometry to customize it, the customized geometry is also stored in the user input lane table. Now, whenever the lane must be regenerated, its geometry will be overridden by the corresponding user input. If the user deletes the lane, a triggers interprets that as a command to return to the default behaviour, and the corresponding row is thus deleted in the user input lane table.

Efficient Multi-user data edit

The work presented in this section has been achieved together with Lionel Atty (SIDT, IGN, all the python development). A proof of concept open source QGIS plugin is available [6].

Collaborative data and Gamification

We observe two major trends in the last five years regarding Geographic Information edition.

6 http://remi-c.github.io/interactive_map_tracking/

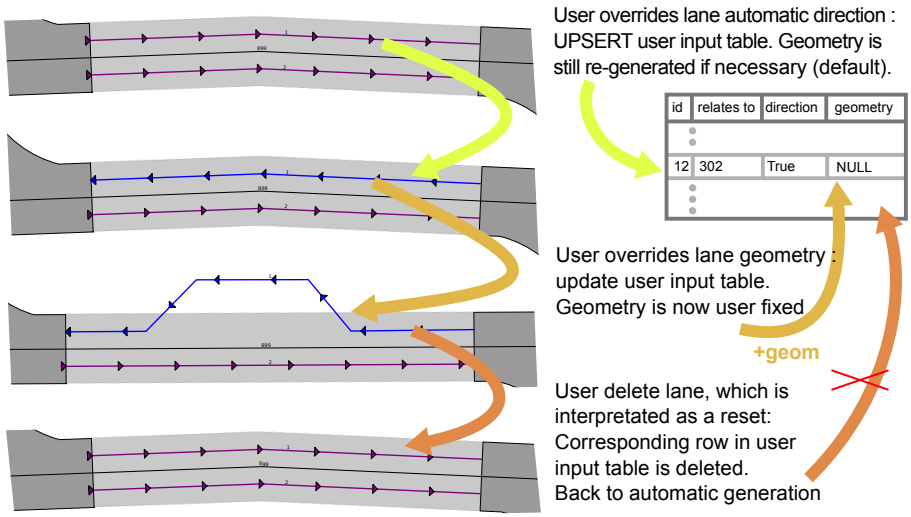


Figure 90: When the user overrides a default behaviour, the parameter is stored in a user input table. Objects overriden do not use default generation anymore thanks to COALESCE. The user can still delete the overriden object to return to the generic behaviour.

COLLABORATIVE EDITING The first trend is toward collaborative editing. The success of projects like OpenStreetMap[7] have put into light the advantages of collaborative editing. Working simultaneously to edit data greatly increase the scaling possibility. Moreover, supporting several users edit may also improve quality, as a mix of advanced and regular users is possible, some user possibly having a quality control role.

SERIOUS GAME The other trend is more pervasive, and concerns gamification (or serious game). Gamification is the integration of game-related elements into a non-game context. For instance earning virtual points, badges, achievement, etc. It has proved to be a powerful incentive that can diminish the subjective effort associated with a task. (See Djaouti, Alvarez, and Jessel, 2011 for a classification of serious game).

We conceptually build on both trends to create a tool helping multi user in-base interaction.

In order to achieve multi user editing capabilities, we need first to use mechanisms so that simultaneous edits of the same data is dealt with by the database (See Chapter 3). That is a safeguard against computer errors. The ability to not crash when several users edit the same data is not sufficient for efficient multi-user editing. Users also need a way to be aware of each others, and most notably of who is working (or has worked) on which area, which is a safeguard against human error.

The first problem was introduced and solved in the Chapter 3. In this section, we propose a solution for the second problem. The proposed methods have been implemented as a QGIS plugin with Lionel Atty.

Better server interaction with auto save and refresh

We introduced in-base interaction concept in 4.3.2. Advantages are numerous, including the possibility to use many clients because all the interaction happens inside the

7 www.openstreetmap.org

database. This interaction concept is based on the fact that when a user edit data or a controller, it will triggers in-base behaviours.

For instance, a user moves a road axis, which triggers the regeneration of associated road surface and intersections, as well as lanes and intersection trajectories. For this interaction to be an efficient human interface, the in-base behaviour should be clear and fast enough to be interactive (less than a second), and most importantly, the user should have a feedback when he performs an action. Based on the proposed architecture, the feedback can only happen when the database received changes on data or controller. Yet, some GIS software like QGIS do not send changes to database unless user specifically asks for it (via a button: saving current change inside a layer). This mechanism is intended to provide a local edition (including revert capabilities) before actually sending the data to database. Based on this, user can not receive any feedback until the changes are send to database. Furthermore, QGIS has no way of knowing that editing data or a controller has changed other layers, whose rendering should also be updated.

As a proof of concept, we create a QGIS plugin to disable the local edition, so that any change on a PostGIS layer is directly sent to the database, and forces refresh of all rendered objects when a change has been sent to database, with a slight delay. That way, all the database trigger-based interactions appear to be interactive.

Easier collaborative editing with user map tracking

Having the database model and interaction being able to deal with more than one user is just one of the necessary steps for efficient multi user work. Users are human, so team work requires adequate processes and tools. We identified one minimal requirement to enable efficient teamwork, which is to be able, at all time, to quickly see what and where other people are working on the map.

We propose an approach inspired by Google Doc[8] collaborative editor, where the editing cursor of each user is highlighted in one dedicated colour for the other users to see. The idea is similarly to display the current and former area of editing of each member of a team.

Each time an user (with this feature activated) browses the map between min_{scale} and max_{scale}, this user screen extend is recorded in a common PostGIS layer. Along the screen map extend are also recorder a unique user id (session name + IP) and time (in ms) . A simple layer style with a random colour per user id allows then to see where each user is working (See Figure 91).

Those screen extents are recorded asynchronously via a stack (LIFO), so as to never slow the editing or reduce interactiveness.

All users record their screen extent in the same PostGIS layer, with an unique identifier and precise time. This allows to create PostGIS views to warn when potential work conflicts occurs. We created two examples of such conflicts. The first is when a user comes back on an area he edited more than 5 minutes before (potential risk of re-editing the same area twice). The second is when two user are editing the same place at roughly (less than 5 minutes) the same time, again potentially risking duplicate work. We stress that all this is only informational (no coercive ability), users still have full control.

8 https://docs.google.com

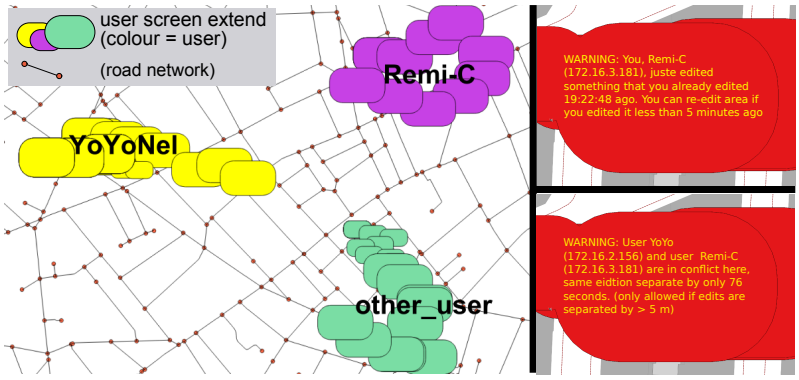


Figure 91: Example of multi-user screen extent tracking. Successive screen extents are recorded through time (oval geometries), a long with a user-id and precise time. Displaying those screen extents immediately informs about who works where. Potential conflicts (one user editing the same area twice or 2 users editing the same area at roughly the same time) are automatically detected and a label appears on the screen.

Collaborative planing and gamification

We presented a tool to allow users where other users are working, that is to facilitate teamwork during edit time. However, in a real life work-flow, some planning occurs before a team edits data for a given area, and some analysis may be performed after the edit is finished.

We propose to use an hexagonal to-do grid to help planning and analysis, as well as introduce a small amount of gamification. Before edit starts, a working area is defined (by several polygons). An in-base interaction (See Section 4.3.4) generates an hexagonal grid covering the defined area. Each hexagon also holds information 'todo' or 'done', 'todo' by default. Hexagonal tiles are red when 'todo', and blue when 'done' When a user map extend is saved, all the hexagons covered by this extend are set to 'done'. The hexagonal map is then a fun way to see what has been done and what remains to do, as tiles change colour.

When edition is finished, the same hexagonal grid can be used as a support to display information, for instance the cumulated edit time.

RESULT

In this section we introduce actual in-base interactions that combine previously introduced patterns (see Section 4.3.3). Those examples are partially extracted from StreetGen (Cura, Perret, and Paparoditis, 2015a), an in-base tool to generate streets. StreetGen models several things that can be edited in different ways. (Videos demonstrations are available for StreetGen (See Fig. 93).)

- The road information (Section 4.4.2), which separates constant width sections and intersections, and model road surface and intersection surface based on curb stone with specific turning radius.
- The traffic information (Section **??**), with lanes and lane interconnection.
- The street objects, which are semantic objects that may be defined relatively to a road axis.

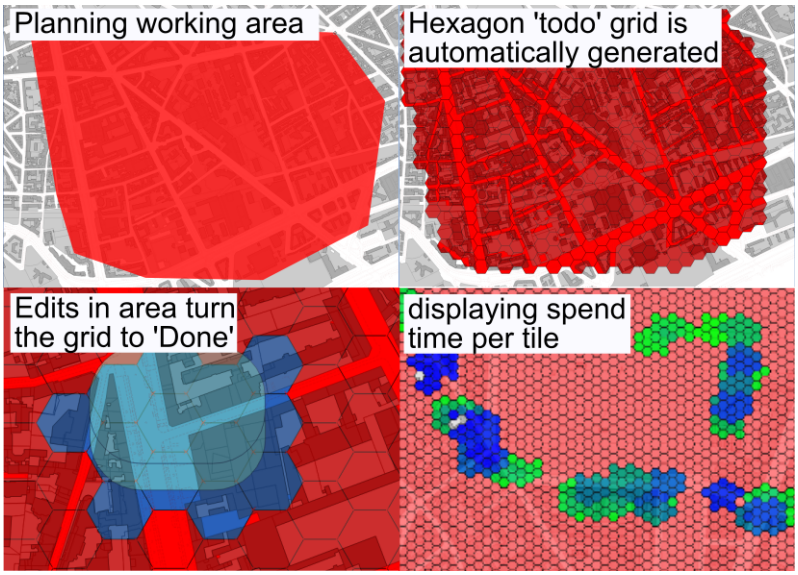


Figure 92: First a working area is defined, which automatically generates an hexagonal 'todo' grid. Then when user work on an area the corresponding hexagons are marked as 'done'. Afterward, the hexagonal grid can be used ot display editing time spent per area.

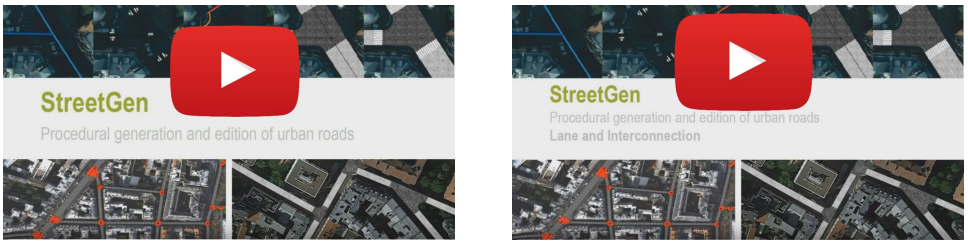


Figure 93: Videos of StreetGen in-base interaction for basic parameters https://youtu.be/rBWZs50wVHg and lanes and interconnections https://youtu.be/yIG_5MBODfo .

In base interaction

We tested the in-base interaction concept with several common open source GIS softwares (several versions of QGIS, OpenJump, GrassGIS). In all cases, edition correctly triggers in base interaction.

Interactive road

StreetGen road model parameters (See fig. 94) are the road axis network topology, road axis geometry, road width, curbstone radius (turning radius) and manual override of intersection limit.

We present in-base interactions divided in two parts. The first part is the core edition, which allows to edit all parameters. The second part is improvement of core edition to create a better user interface.

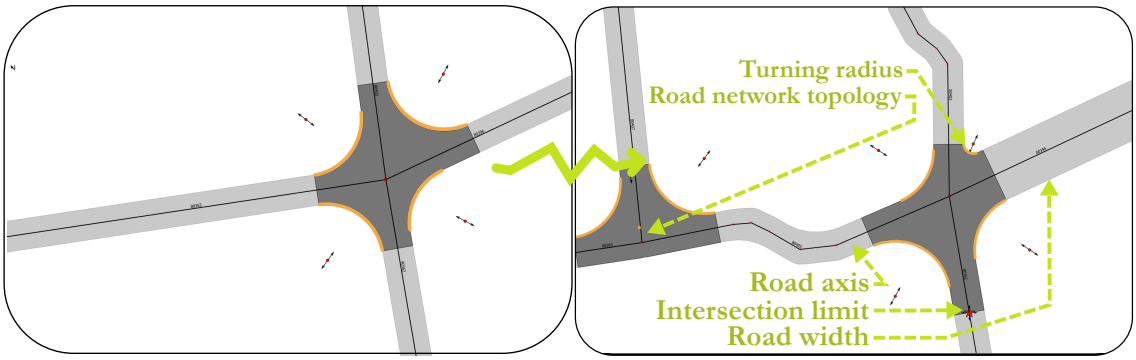


Figure 94: StreetGen road model parameters, each can me modified using in-base interaction.

Road editing

EDITING POSTGIS TOPOLOGY NETWORK The very basis of StreetGen modeling is a road axis network that uses PostGIS Topology. Interactive topology edition is complicated, especially if topology is semantized. The problem stems from the necessary interpretation of user action to transcribe it into topologically valid operations. We implement this in-base interaction using the "Proxy view" design, so as to provide a safe and dedicated interface. We add two views : 'edit-node' and 'edit-edge'. We propose a proof of concept free and open source [9].

Lets take an example where the topology only contains two nodes and one edge (a line) between the nodes. The user creates a new node that is close to the middle of the line, but not exactly on the line. This behaviour has to be interpreted has "I clicked close to the line, but in fact I meant on the line", thus this node has to be snapped to the line and the edge split into two parts, with relevant topological information updated, as well as semantic'.

In more complex edition cases the expected behaviour might not be so well defined. Therefore, when we create in-base interactions to edit a postgis topology, we purposely limit the possible user actions with explicit error messages. We limit the interaction so that in any case we can use the postgis topology API safely. The main limitation is that except in obvious case, no edge split automatically occurs. See Fig. 95 for example of user interaction.

EDITING STREETGEN ROAD AXIS NETWORK Postgis topology interactive editing only changes topology. Other triggers are necessary to adapt it to the StreetGen data model, and ultimately regenerate axis that have been created/updated. In particular, deleted or created edges will have an impact on all of the StreetGen data model tables. This changes are propagated with a mix of trigger and using postgres foreign key (for delete).

EDITING OF ROAD AXIS Road axis can be edited both for the geometry and for attributes, such as road width and number of lane. We use the proxy view to separate user

9 https://github.com/Remi-C/postgis_topology_edit

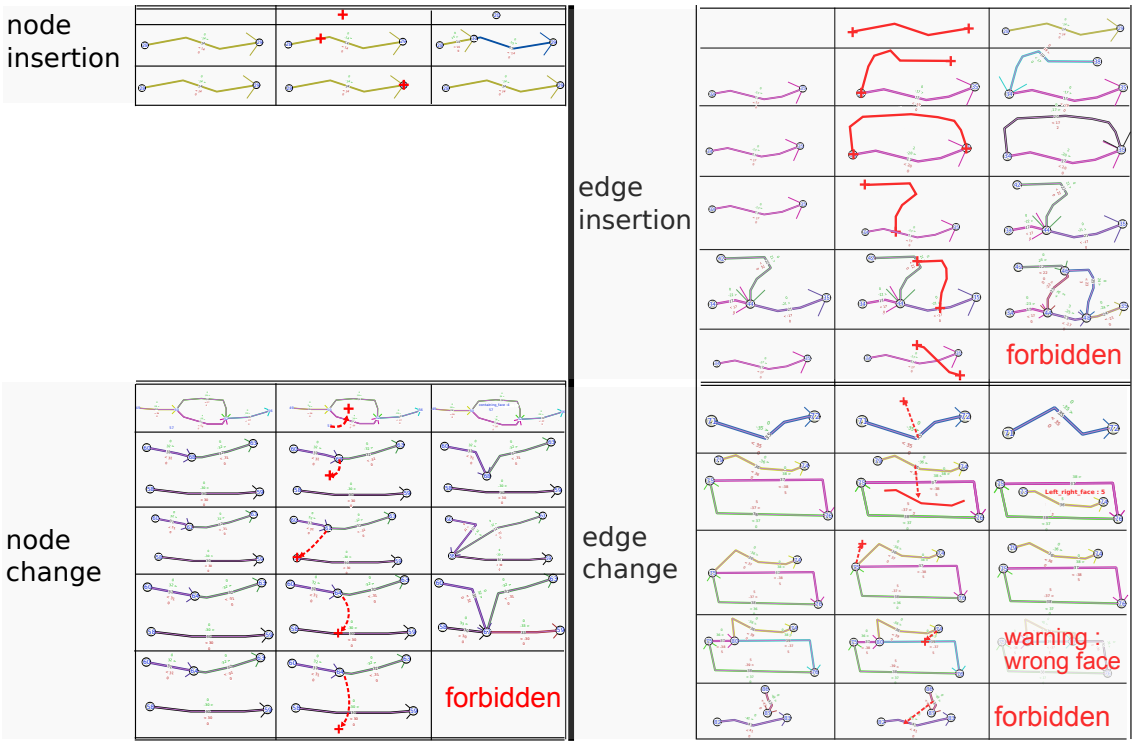


Figure 95: Example behaviour of in base interactive topology edit, through two views, 'edit-node' and 'edit-edge'. Ambiguous user inputs are avoided.

input from automated modifications. In fact, any change of input data like this simply triggers a relaunch of StreetGen on the concerned elements, which unify processing.

User can edit radius by typing a new value, or use the indirect "geometric controller" as seen in Section 4.3.3.4.

EDITING OF INTERSECTION LIMIT As seen in 4.3.3.3, intersection limit uses a combination of indirect "geometric controller", "proxy view", and "sotring user choice".

ADVANCED ROAD WIDTH EDITING Default way to edit a road width is to change the 'width' value of the road axis attribute, which is a lot of clicks and a waste of time, as most likely the user does not know the correct value, and will have to try several widths.

Instead of that, we propose to use an indirect "geometric controller" to streamline road width edition. The idea is to indirectly provide road width by indicating where the sidewalk is. Road width is then automatically extracted from the potential sidewalk position by first assigning each border points to a road section using section surface (which can be done efficiently with geospatial indexes), then compute new width by taking the median value to the distance to the relevant road axis. Road width is then updated with the new value, and the relevant road is re-generated.

We present in-base interactions divided in two parts. The first part is the core edition, which allows to edit all parameters. The second part is improvement of core edition to create a better user interface.

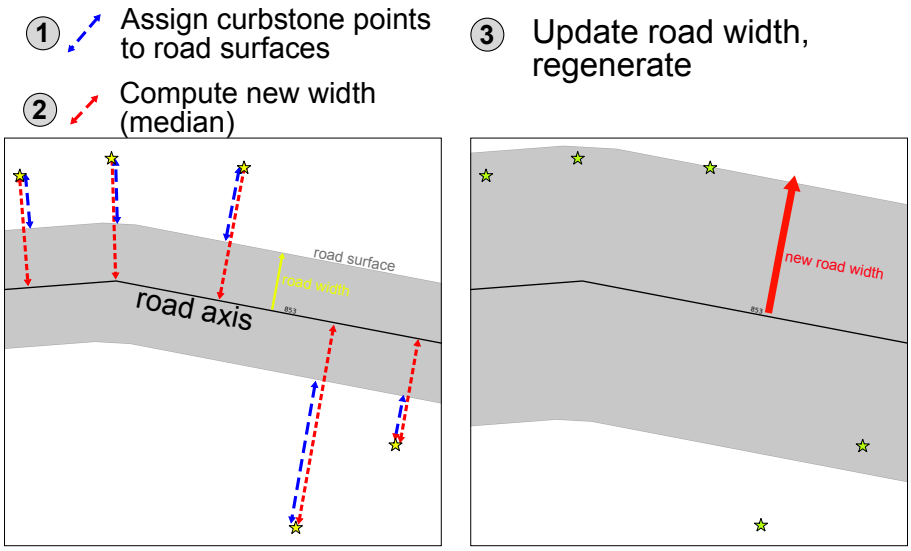


Figure 96: Road width editing is extremely facilitated by the use of an indirect geometric controller. The user simply add points on the cornerstone and the road width is automatically updated, instead of guessing a width and typing it.

In an informal experiment, we try to find the correct width of a street with and without the indirect controller. Without controller, it takes a dozen tries to find the correct width with less than 0.1 metre error (30s). With controller, it's just one click (2 seconds). The advantage of the controller interface is even more obvious when road axis geometries are adjusted afterward. Indeed, if road axis is not well centred, the road width must be adjusted, as opposite to the controller version, where the road width is automatically recomputed when the road axis geometry is changed.

Interactive traffic

StreetGen also generates basic traffic information, such as lane geometry and direction, and interconnections between lanes in intersections. Both lane and interconnection use a combination of "Proxy view" and "user choice". Furthermore, interconnection also uses an indirect "geometric controller" as control point of the Bezier curve.

lane editing

As seen in Section 4.3.3.5, the user can edit each lane direction and geometry. When the user edits direction or geometry, he steps outside of automated generation (Automated results will be overridden by user edits). However, the user can return to automated generation by deleting the lane, which is then interpreted as a delete on user override rather than a real delete of the lane. Lane number is also editable through another "Proxy view" on road axis.

interconnection editing

Interconnection have a more complex behaviour. In lane case, a lane always exists, because the number of lanes is a parameter. For interconnection, the user needs a way to

convey the information that an interconnection between two lanes might not be authorized. For instance, at a given intersection it might physically possible to turn left, but it is forbidden by law.

This information is stored in a boolean. The user does not set it directly, but instead deletes the interconnection geometry. If interconnection was not overridden, then the interconnection is marked as not allowed. Else, user parameters are deleted as for the lane case.

A particularity of interconnection is the use of Bezier curves to model trajectories (See Fig. 97).

This curve is controlled by classical control points that are stored in the same table as the interconnection trajectory (default control points are implicit).

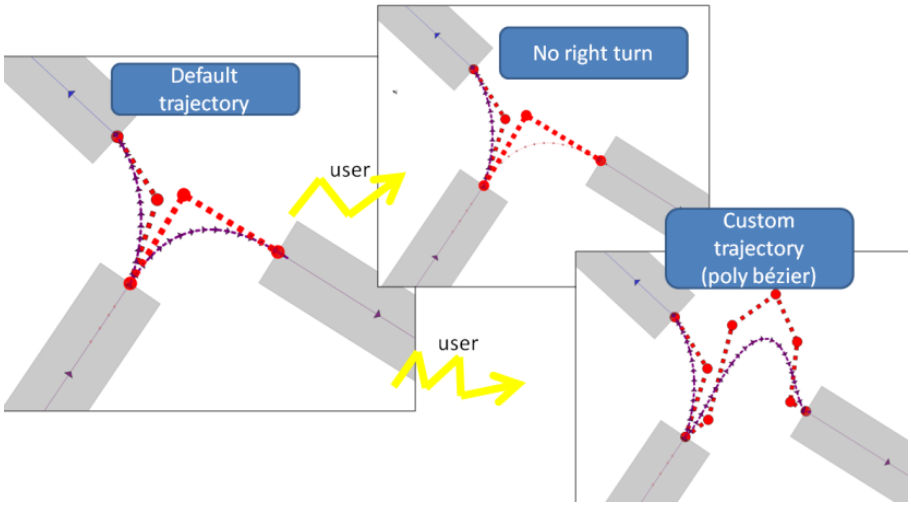


Figure 97: User editing interconnection, both for customized trajectory with bezier curve and possibility to use this trajectory.

Interactive Street Objects

Generic street objects

Generic street objects are semantic points that can be positioned relatively to a street axis. If this is the case, object position is defined by an curvilinear abscissa and distance to street axis (or to cornerstone). We stress that relatively positioned object must still have a synchronised absolute position so as to be correctly displayed in GIS.

Object orientation can be similarly absolute or relative to a street axis. When a street axis is affected (change of geometry, of width, or change of topology), all the relevant street objects are updated so as to have coherent relative and absolute positioning (and orientation). Object positioning types may be overridden. For instance, if an object is positioned relatively to a street axis, and this street axis is deleted (topology change), this object must be switched to absolute positioning.

This mechanism warranties that objects are always coherent. We also use "Proxy view" so that user can interactively create/edit/delete street object. At object creation, user

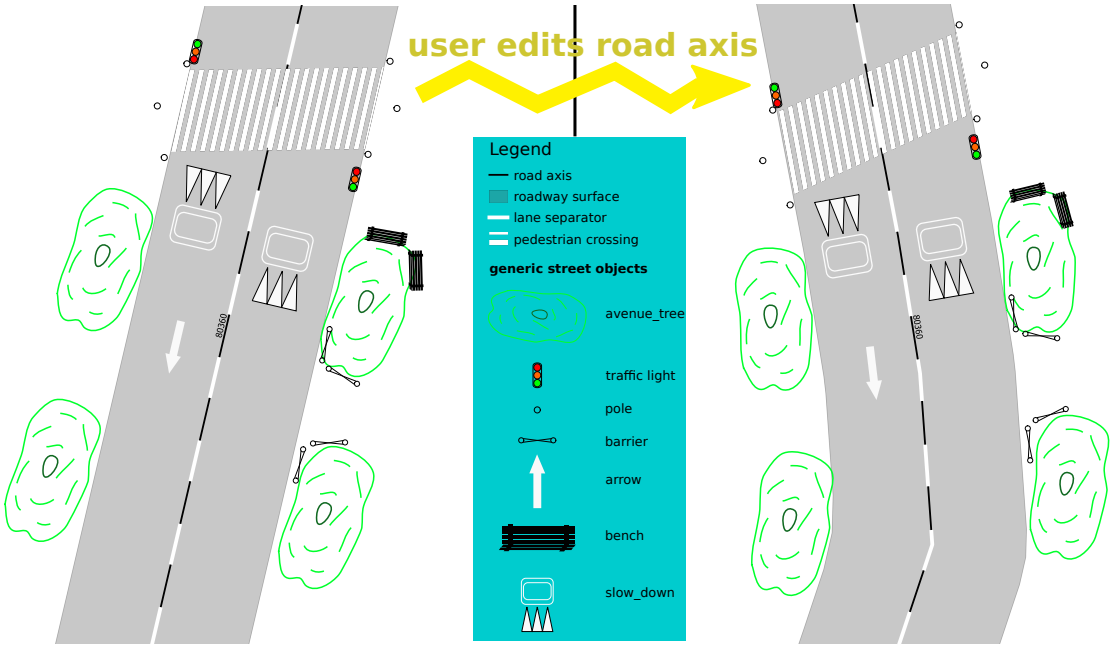


Figure 98: Street objects can be defined relatively to streets. In this case, a change on street automatically triggers the re-computation of the absolute object position and orientation.

makes a choice between relative and absolute positioning (and orientation). In case of relative positioning, the reference can be street axis or side-walk.

Interaction handling is very complex when object are in a relative position. The first level of complexity comes from the necessity to synchronise relative and absolute positioning, knowing that the user can change both, and that those changes must always be transcribed into relative positioning. For example, an object is defined relatively to a street axis. The user moves the object in a GIS software (thus changing the object absolute position). Then trigger interprets this move as a need to update relative positioning based on new absolute positioning. Then a new absolute positioning is generated based on new relative positioning. In this, street object becomes its own "geometric controller ".

The second level of complexity comes from implicit reference handling. Indeed user choosing relative positioning never explicitly indicates to which street axis the object refers. Instead, this axis is automatically found (closest one at creation) and updated (if an axis is split for instance, or if user moves the object very far from the street). For instance if the user moves an object from one street to another, the relative positioning gets updated and references the new street axis.

Specialized street objects

We illustrated the possibility of specialised objects with a proof of concept example about pedestrian crossing.

Specialised objects add a layer of automation on top of regular street objects. For instance, pedestrian crossing creation is done via a 'proxy view' strategy. User create a polygon roughly representing the pedestrian crossing (possibly using only 3 points).

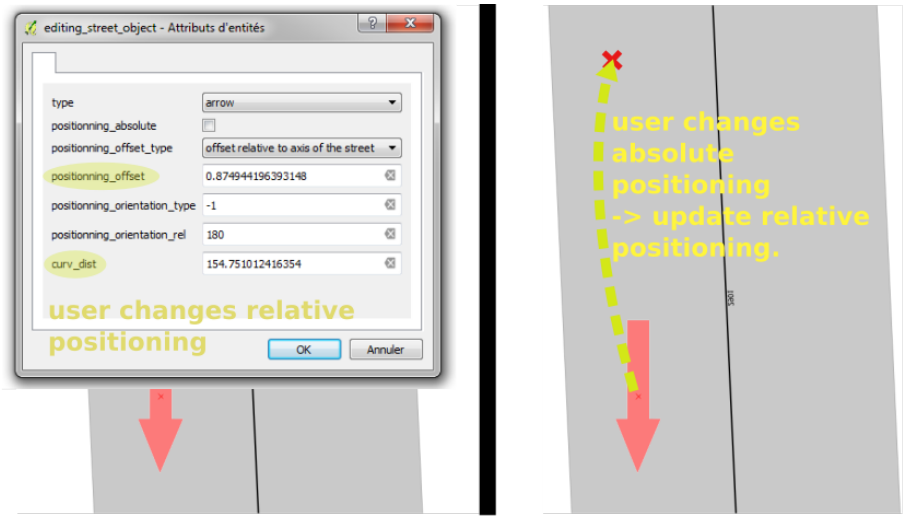


Figure 99: Street objects edition can be done through the change of attributes (here, with QGIS). Simply moving the object also automatically updates relative positioning information.

This polygon is then analysed to extract pedestrian crossing parameters (width and orientation). We could not find in the literature a method to robustly (regarding the number of points and points repartition) fit a parallelogramoid (both side of the road may be polylines). Therefore, we propose a simple one: each segment of the polygon envelop votes for an orientation (weighted by segment length). Final orientation is the average of the votes. The pedestrian crossing width is determined by separating segment into left and right of the road axis. Each side determines a width, the final width is the average width of both sides.

In fact, finding the best pedestrian crossing model adapted to user inputs is already inverse procedural modelling.

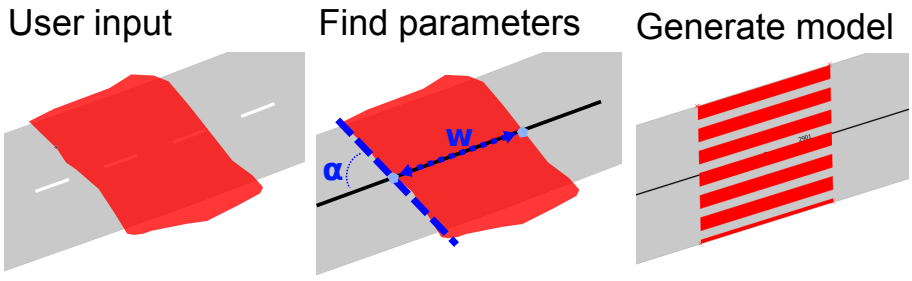


Figure 100: Specialised object pedestrian crossing creation is greatly facilitated by automatic parameter extraction and generation.

When a user modifies a pedestrian crossing geometry, its parameters are recomputed. This way, one graphic controller allows to control all pedestrian crossing parameters (position, orientation, width). Again, this allows for easy parameter changes via "graphic control".

Efficient Multi-user data edit

Better server interaction with auto save and refresh

The auto save and refresh plugin is not necessary per se, provided the frequent use convenient save and refresh short cuts. However it adds a great deal of comfort and reduces the number of clicks.

DISCUSSION

In this section we discuss elements of method (4.3) and result (4.4) sections. We start by analysing need of interaction for procedural modelling, and proposed design patterns. We examine then how the proposed method to facilitate multi-user work perform.

The next part of discussion is dedicated to interaction in StreetGen, with interactive road, traffic and objects modelling.

In base interaction for procedural modelling

We stress that the results that can be obtained by our method (interactive procedural modelling) are limited by the modelling capabilities of the procedural tool. Our simple road model (fixed width + intersection) can not model all existing roads. Prominently, some road have varying width. Similarly, our model can not generate all type of cornerstones, for instance cornerstones using two successive radius, or chamfered.

We propose to amove the interaction from client to database. While it brings numerous advantages, it is ill suited for very complex interactions, where dedicated Human Machine Interface (HMI) would be more appropriate. Indeed, in base interaction can happen only *after* an edit occurred, which prevents any HMI scenario where the HMI proposes solutions *before* edit is done. Yet those interaction fall in the significant Guided Design trend. When the complexity of interaction increases, the current PostgreSQL trigger framework also becomes a serious limitation. Most popular User Interfaces (UI) are based on signals (for instance QT[10]), which can be rudimentary mimicked by PostgreSQL triggers. However, triggers offer almost no modern control, and, as such the difference with modern UI is similar to the difference between assembly languages and modern object oriented languages. Therefore, in base interaction scales badly in terms of code complexity, generality and maintenance. As such, in base interaction should be limited to straightforward cases, and not be pushed too far.

Different in-base interaction types

We propose several patterns to facilitate in-base interaction, yet, the distinction between patterns is quite artificial, and real use cases tend to blend all patterns.

Using controller and/or proxy view necessarily increase the database server workload. For a "Proxy view" strategy, the choice between a "VIEW", a "TABLE", or even a "MATERIALIZED VIEW" may vary a lot depending on the load, quantity of data, complexity of code maintenance, etc. In this article we only explore triggers for in base

10 www.qt.io/

interaction. Yet, databases also have powerful rule systems that could be used for basic interaction.

As a perspective, storing user choices is a first step, but more advanced features could be attained, like storing user choices and archiving it, so as to have access to former user choices, rather than delete/overwrite.

Efficient Multi-user data edit

An obvious limitation of the auto save and refresh is to disable local undo/redo control. It also breaks the concept of in-base interaction as it creates client-side code. The user map tracking is fuzzy by nature, as the screen map extent is registered each time the user changes it, regardless if an edition occurred or not. This can not be avoided, as sometime quality control (mostly not editing things, but checking parts of the map) is as important as edit, and should also be tracked. Indeed, two users performing a check on data could easily check the same area without being aware if not using the plugin. The gamification concept could be pushed much further, with virtual points, awards, etc. More importantly, a real edit work flow would benefit from more advanced tools with user having multiple roles (editor, checker, manager, etc.). All the role interactions could be helped by plugin, and happen in base with the hexagonal grid support. For instance, a team leader could assign different areas to be reviewed to his colleagues. After delivery, a client finding a problem could mark the relevant hexagons, so the that edit team has easy and immediate notification of erroneous area. We stress that although not limited in theory, we never tested the plugin with more than 3-4 users.

Interactive road

Road editing is seriously limited by the topological road axis network editing. Indeed, our interactive topology editing may lead to incoherent in the implicit faces of the topology. This is an implementation limitation rather than a conceptual limitation. Currently there is no way to split many edges at the same time, to introduce a road axis cutting Paris in half for instance. We noticed however that this interactive topology edition is very useful compared to the alternative, which is to recompute the whole topology from scratch for each change. PostGis Topology is not fast, building topology for paris street is several minutes. In some case, we would benefit from higher level operation, like replace several small intersection with one roundabout for instance.

We also noticed that introducing geometric controller for turning radius and road width is extremely helpful, with speed gains about one order of magnitude, and edit much more agreeable.

Interactive traffic

Users can edit lane and interconnections. Lane direction editing could be more effective, maybe using a geometric controller. Users have to edit a field 'direction', which is not handy, especially when several lanes could be edited at once. When users delete interconnections, it actually sets the interconnection trajectory as forbidden. This behaviour greatly speeds editing, because several interconnections can be selected and deleted at

once. However, by default all possible interconnection are authorized, which creates a great number of interconnections in intersections with many lanes. Interaction would be much more efficient if we could use some heuristics that would connect lanes more conservatively. Bezier curves are great for ease of control, but rather inaccurate when it comes to actual vehicle trajectory.

Interactive Street Objects

The street object interface is especially useful as it allows to somehow compensate limitation of road model. For instance, StreetGen road model does not consider parking spaces, which is a strong limitation in Paris where parking spaces are omnipresent, and especially meaningful for urbanism. Yet those parking spaces could be modelled as street objects, using an adhoc object specialisation similarly to pedestrian crossing. We presented a specialisation for an object which is a surface by nature (pedestrian crossing), yet many street objects are also linear by nature (like some markings and barrier). Street objects were only tested at street scale.

Moreover, we only presented interactive editing of street objects, and not large scale generation, with advanced patterns and rules. Good examples of adequate complexity can be found in shape grammars designed for city generation.

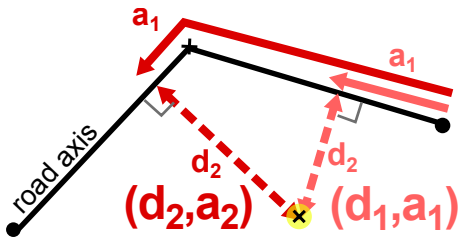


Figure 101: A unique absolute position can correspond to several relative positioning, which theoretically limits relative positioning setting through absolute geometry.

There is a very fundamental limitation to switch between absolute and relative positioning like we do (See Fig. 101). Basically, going from relative to absolute positioning is not a bijection, so, in some cases there is no inverse function (going back). So, in some cases a couple (curvilinear abscissa, distance to axis) may not be settable through geometric proxy. This problem also affects the altimetry example we gave (See Fig. 88). However, when special cases require it, it is still possible to set the relative positioning manually.

Best of 2D and 3D world for edition

The proposed method is based on common GIS softwares, which represent and deal with data in 2D. The 2D view (map view) has obvious advantages for edition: it is simple, clear, and edition can be efficiently performed with usual interactive devices (mouse). Yet, the 2D view is sometimes confusing, especially for objects like street signs that might become invisible in 2D. Our brain is also extremely good at understanding 3D scenes. In this optic, we could propose a mixed 2D-3D edition to get the better of both worlds.

We explored this idea with a prototype (work performed by Lionel Atty), where we use QGIS as he main 2D edition software. We create a plugin containing a web browser, then display ITowns[11], a WebGL application able to show streetview, street Lidar, and perfom measures and edition. Both are synchronized, so that edits in 3D are also displayed in 2D, and so that 3D camera position and orientation is also controlled and displayed in 2D. The 3D streetview gives exemplary context awareness, and can be used to perform precise edition, while the 2D view gives good overview and fast navigation, and can be used for classic edit.

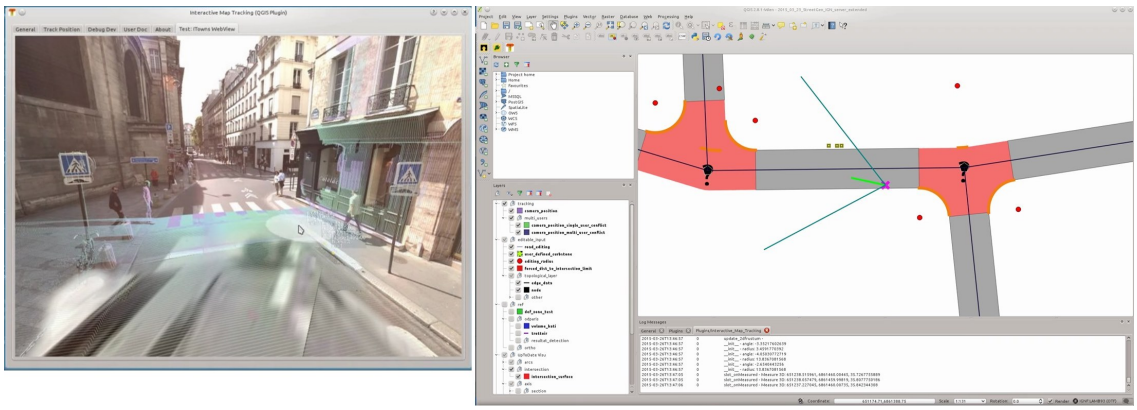


Figure 102: Coupled 3D (left, ITowns) and 2D (right, QGIS) visualisation and edit, providing clear and fast edit (2D) with advanced view and 3D capabilities (3D).

CONCLUSION

In this chapter we proposed a new paradigm for custom user interaction with GIS software, where interaction handling is moved from GIS softwares to the database. In this paradigm, GIS software simply modify geometries and attributes of database layers, and those changes are used by the database to perform automated tasks. In the most basic form, this automated interaction can be used to check changes, for instance rounding coordinates. The database can intercept the change and adapt it, for instance to automatically simplify a polygon. For more complex interaction, we demonstrated the use of geometric controller, which are conceptually close to UI controller such as sliders, but are made of geometries with attributes. Such geometric controllers can exempt a user from using a form, thus being an order of magnitude faster. We demonstrated those capabilities with several examples of various complexity, including the interactive editing capabilities of StreetGen, an in base procedural street generator tool.

Last in base interaction can also be taken one step further and be leveraged to help team work. In particular, work planing is possible before editing, work analysis is possible after the edition is completed (quality), and the edition can even be enhanced by introducing gamification elements.

11 github.com/iTowns/

5

INVERSE PROCEDURAL MODELING

To reconstruct actual streets, a street model (Chap. 3) has to be adapted so it matches actual streets. For many use cases a manual modification may be sufficient (Chap. 4). Yet, for large scale reconstruction, we believe that a (semi) automated process using observation(with the support of observations (Chap. 2) is necessary. Directly fitting a street model to raw urban features would be intractable and extremely difficult, as the street model contains objects of various type (road axis network, lane network, road surface, street objects, etc.). Moreover, we established that all elements of the street model seem to be organised around the road surface. For instance, a white rectangle marking would have totally different role (and thus be a totally different urban feature) depending on its context: on the road surface or outside.

Therefore we chose to first focus on reconstructing road surface, which could then be used to fit other aspects of street model more easily, and ultimately lead to full street reconstruction.

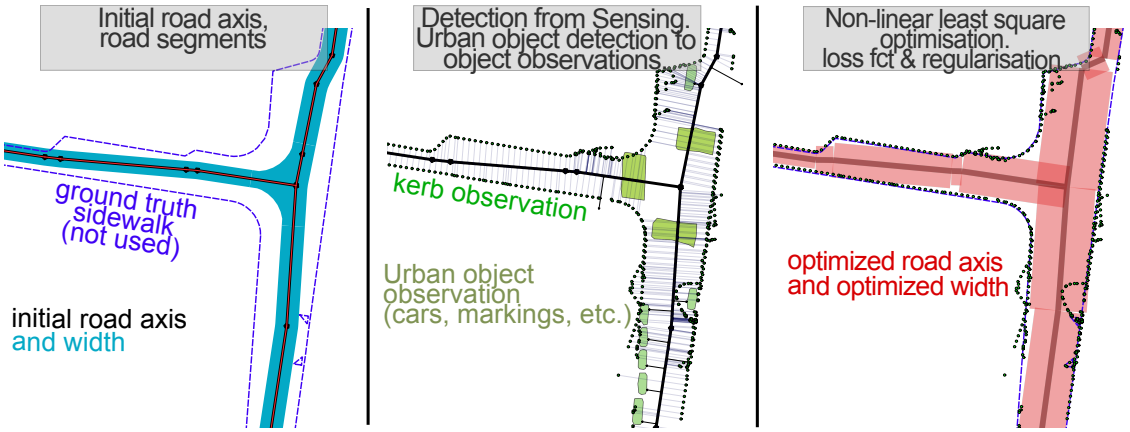


Figure 103: Approximate road axis network and road width are available, forming a basic road modelling. Various sensing methods produce urban feature detections which are processed into consolidated observations, and assigned to road axis segments. A robust non linear least square optimisation then fits the road modelling to observations. The result is much closer to ground truth, even when the road model is too simple for the actual road configuration (varying width, asymmetric changes, curves). The user can further input observations if necessary.

Cities are structured by roads. Having up to date and detailed maps of these is thus an important challenge for urban planing, civil engineering and transportation. Those maps are traditionally created manually, which represents a massive amount of work, and may discard recent or temporary changes. For these reasons, automated map building has been a long time goal, either for road network reconstruction or for local road surface reconstruction from low level observations. In this work, we navigate between these two goals. Starting from an approximate road axis (+ width) network as a simple road modelling, we propose to use observations of street features and optimisation to improve the coarse model. Observations are generic, and as such, can be derived from various data, such as aerial images, street images and street Lidar, other GIS data, and complementary user input.

Starting from an initial road modelling which is at a median distance of 1.5m from the sidewalk ground-truth, our method has the potential to robustly optimise the road modelling so the median distance reaches 0.45m fully automatically, with better results possible using user inputs and/or more precise observations. The robust non linear least square optimisation used is extremely fast, with computing time from few minutes (whole Paris) to less than a second for a few blocks.

The proposed approach is simple, very fast and produces a credible road model. These promising results open the way to various applications, such as integration in an interactive framework, or integration in a more powerful optimisation method, which would be able to further segment road network and use more complex road model.

Problem

Paris is a large city, with thousands of kilometres of streets. Unlike highways, the vast majority of roads in those streets do not follow strict design guidelines due to historical reasons (they also pre-date civil engineering guidelines, which have also evolved anyway). Yet, an up-to-date precise map of those roads is essential for many applications, like city planning, urbanism, traffic analysis, autonomous driving or simply help wheelchairs or stroller users navigate urban space. Streets are also changing frequently, with very frequent public work and planning efforts, as well as effects from other civil works and maintenance. As a testimony, the ground truth data we use dates from 2011, however, actual sensing from 2014 showed that a part of the side-walks had been modified, and we only looked at a small area of a few blocks (See Fig. 119 on page 171, right)!

Manually creating and updating those road modelling is extremely time consuming, and we wonder how much time and efforts went into the 1860 Paris plan that include side-walks (Fig. 104). Thus, a mostly automatic method is needed to create/update

Figure 104: Plan parcellaire municipal de Paris, 1860. Copyright Archive de Paris.

these road modelling. Automatic methods need data: actual observations that can be leveraged to find an adequate road modelling. The ways to observe roads are numerous, be it through street images or street Lidar, aerial images, legacy GIS database or digitized legacy maps. All those observations sources should be usable by the method, as each could suit a particular situation. For instance high buildings reduce the interest of aerial images, but have no impact on street Lidar. Yet street Lidar data is unlikely to be available outside of major cities.

Thankfully, such a road modelling is usually not to be created from scratch. At least, some kind of road axis network is usually available, either from a database, or reconstructed from sensing data (aerial image/Lidar, vehicle trajectory, phone tracking, etc.), although reconstructing a road axis network is already a very challenging task in itself.

Of course observations from sensing are not perfect, especially when several sensing sources are mixed. Therefore, to model a road is in fact akin to finding the optimal road modelling that fits those observations. A road model with parameters must be defined, then, a suitable optimisation method can be used to find the optimal values of these

parameters. The observation may be erroneous, and so can be the road axis network. Finding the optimal road modelling may then not only involve finding the optimal parameter values of the road model, but also to find the optimal number of parameters, as well as the observations that should be used.

Moreover, urban roads are sometimes so complex that some form of user interaction is most likely necessary.

Related work

Road surface reconstruction is a popular topic with widely varying methods depending on the input data, the complexity of expected result, and available computing time.

OBSERVATION ORIENTED (SENSING) Many methods focus on fast methods producing low complexity road models (see Bar Hillel et al., 2012 for a state of the art). Those bottom up methods can use street Lidar as in Zhang, 2010, street videos (Zhang et al., 2009), or rasterized Lidar (Yang, Fang, and Li, 2013). In those cases, the goal is more to classify which pixels or points pertain to road, rather than reconstruct a high level road model.

MODEL ORIENTED (GIS) On the other side of the spectrum, numerous methods focus on rebuilding the main component of a road model: the road axis. Reconstructing road axis involves topology, and can be performed with various data, such as aerial image (Montoya-Zegarra et al., 2014), Lidar (see Quackenbush, Im, and Zuo, 2013 state of the art) or vehicle traces (GPS data, see Ahmed et al., 2014).

Such methods focus on network reconstruction, and usually do not consider road geometry, with the exception of Zhang, Thiemann, and Sester, 2010 that reconstruct a road network from GPS traces and also estimate road width, and Clode et al., 2007, which use aerial lidar data to reconstruct a road network topology along with the border of the road (width is varying along the road).

FUSION Taking advantage of both observation-oriented and model-oriented methods, some methods fuse low level observation with high level road network. For instance Hatger and Brenner, 2003 initialise a road segmentation method starting with a road axis network database, and using aerial Lidar. Road width and slope is then extracted from the segmentation, but the road axis are constant. Similarly, Boyko and Funkhouser, 2011 start from an approximate urban road network and use it to initialise a road surface seeking method (snake) based on Lidar raster. The road surface is the final result.

Both these methods directly use low level data (Lidar) to segment road surface.

In this work we take a different approach. We start from a higher level road model (road network and road width) to which we associate semantic observations extracted from low level detections (using aerial Images, street Lidar and images, other GIS data, and user input). We then optimise the road model so it better fits the observations. The result is a fitted high level road model.

Approach

Road modelling as a whole is complex, especially if we want to be able to use diverse observation type and do so fast enough to allow user interaction. Moreover, even a superficial look at Paris streets outlines numerous odd roads configurations that would be complex to extensively integrate into a road model, if possible at all. A complex model and diverse uncertain observations make a dangerous mix. Therefore, we chose to model roads in a very simple yet flexible way: a road axis network (composed of road segments) with a road width for each segment. Such a model is a simple basis upon which more complex cases can be added when necessary.

Then, we break the optimisation problem of finding the optimal road axis and road width given a set of observations in two parts. The first part aims at robustly finding optimal values of this road model parameters given observations, road axis segments and associated width. We consider it to be the core and maybe the task that potentially needs the less user intervention. As such, it should be fast and robust, yielding good results in most situations. Some areas may need to be manually corrected when the road model is not sufficient to handle a complex situation.

The second part aims at finding the optimal number of road segments, and which observations affect which segments. This involves splitting/merging/creating/deleting road segments, and removing observations. In our work this part is performed manually. It could be automatized using a powerful optimisation framework, which we leave for future work.

Plan

The rest of this chapter is structured as follows: In Section 5.3, we justify the choice of our road model and optimisation method. Then, we explain how we create observations from sensing detection and how we use them in the optimisation process. In Section 5.4, we show results of observation creation and usage in the optimisation, with optimisation results for various observations and situations. In Section 5.5 we discuss results, limitations and perspectives.

METHOD

Choosing a model and optimisation method

Context

It seems that the way a problem is modelled strongly depends on the optimisation method that will be used. Therefore, before exposing our modelling and optimisation method, we explain our goal and the context.

We suppose we have access to a road axis network with associated estimated road width. As explained in Chapter 3, this hypothesis is not a reducing one, as those data are fairly accessible and when not available can be estimated from various inputs (aerial images, LIDAR, GPS, etc.).

These road axis and road width are only approximate, and lead to a first credible roadway modelling solution, but the goal is to better estimate these values. For this,

we suppose we also have access to observations related to road or street objects (in a broad interpretation), for instance road markings, kerbs (separation between roadway and side-walk), public lights, etc. Again, this hypothesis is not very limiting, as many methods exist to extract those observations from images, aerial view, Lidar, digitized maps, etc.
We interpret an observation as the probable presence of an object, as such each observation is completed by a confidence measure. Each object type has a defined behaviour regarding road surface. For instance, pedestrian crossings are expected to be *within* the roadway (road surface). From an optimisation perspective, the goal is then to find the best parameters of the model to fit the observations.

Optimisation requirement and choice

Following the capabilities of StreetGen (Chapter 3), we seek a method that can work seamlessly from street to city scale. Moreover, for reasons exposed in Chapter 4, and because observations may be missing entirely in some place, or be noisy, the method must allow seamless user input integration, as well as user override. More than user input integration, the method also has to be fast enough to be interactive at street scale so that user interaction is possible. Lastly, observations can concern many different street objects, and take many forms depending on the sensing method. Therefore, the optimisation method has to be generic enough to integrate many types of information about urban objects.

We considered optimising on loop of kerbs around a city block, but we rejected it because the street network could be incomplete, and because optimising one isolated street should be possible.

We choose to formalize the problem as a mechanical problem, where observations exercise forces over the road axis, and road axe are also subject to forces that resist changes so the final solution is not too far from the initial one.

Modelling the problem

Model

We consider the road axis (polylines constituted of road segments) as a set of connected points. To each road segment is associated a road width value (parameter / variable). Segment and width together implicitly describe a simple roadway (road surface), that is a set of rectangles. The variables (i.e. the only values that will be changed by optimisation, which we will call parameters) of the optimisation are then 3D points (3 scalars per points) constituting the road axis vertex, and for each road segment, the road width (on scalar per segment). See Figure 105 and equation 2 on page 155.

Observations are semantic 2D geometries (points, polylines, polygons, geometry collections) associated with a confidence.

The semantic of street objects is their type (about 100 classes, such as car, street furniture, building, etc.). Each street object class is associated with an overall precision, confidence, class weight, as well as the expected class behaviour regarding road way. We consider those as settings of the optimisation, as those depend on data and knowledge on road. Each class has an expected position regarding roadway, which is either *within* the roadway area or *outside*. The expected position can also be defined as in the

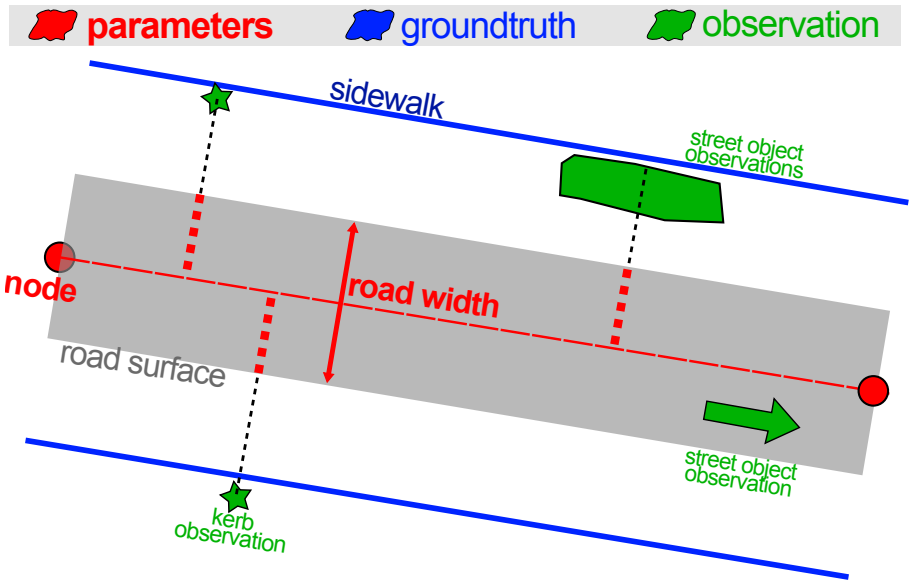


Figure 105: A roadway is modelled with connected nodes and width, which implicitly describes a crude road surface (a rectangles). Observations are defined regarding the implicit road surface.

border of the road area (in or out of the road), or undefined. Furthermore, if the class is defined as being on the border of the roadway, it has an additional expected distance to the border (See Fig. 106).

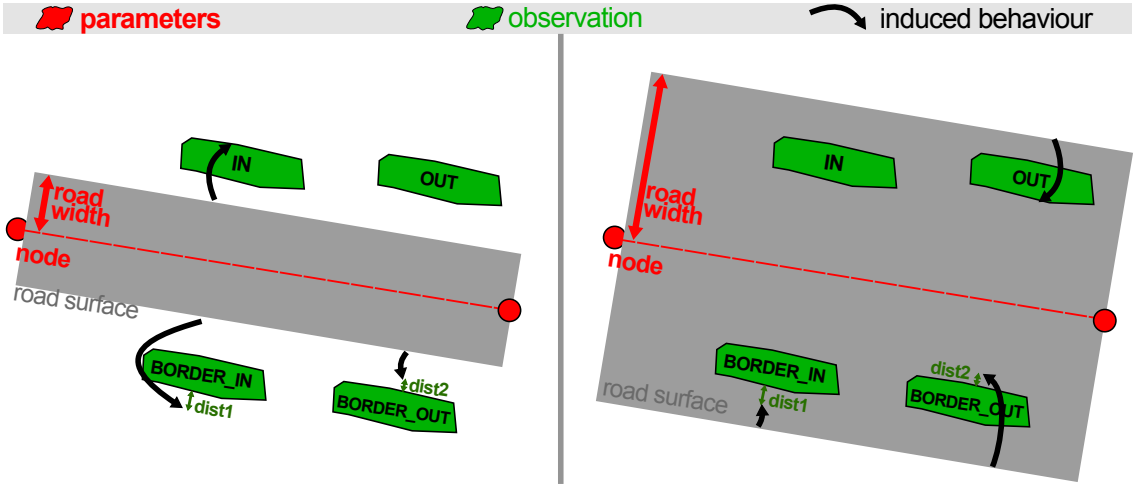


Figure 106: Each expected observation position is defined regarding the implicit road surface, with four options: In, Out, Border_in, Border_out. When Border is used, a distance to the road surface limit can be defined (dist1, dist2 here).

Optimisation method

We choose to use a robust non-linear least square optimisation to find the optimal road axis and road width. We use a generic open source tool to do so (Agarwal, Mierle,

and Others, 2016). The main reasons to choose this family of optimisation is that given observations and parameters (a road axis and road width), we can significantly measure how well the parameters fit the observations, and more importantly, we can explicitly compute changes on parameters to improve fitting.

We consider the problem similarly to a mechanical problem, where all observations generate forces that can be modelled as geometric vectors. This kind of problem can be successfully solved with non linear least square.

Another reason to use non-linear least square is that it is extremely fast, which is one of our requirement for user interaction. So, given an initial road segment network constituted of
n, a set of nodes,
w a set of width relevant to segment (pair of nodes),
o a set of observations
F_o a set of Forces induced by observations and
F_r a set of regularisation forces,
we look for the solution of the optimisation problem S, so that

$$\begin{aligned} S(n,w) &= \arg\min_{n,w}(|F_o(o,n,w) + F_r(n,w)|^2) \\ F_o(o,n,w) &= F_{kerb}(o,n,w) + F_{object}(o,n,w) + F_{direction}(o,n) \\ F_r(n,w) &= F_{position}(n) + F_{length}(n) + F_{width}(w) + F_{angle}(n) \end{aligned} \tag{2}$$

Forces generated by observations (detailed in Sec. 5.3.4.1 on page 162):
$F_{kerb}(o,n,w)$ is generated by the kerb points,
$F_{object}(o,n,w)$ is generated by the street objects surface,
$F_{direction}(o,n)$ is a target road segment angle computed from kerb points.

Forces that regulate the results (detailed in Sec. 5.3.4.2 on page 164):
$F_{position}(n)$ limits the nodes toward their initial position,
$F_{length}(n)$ limits the road segment toward their initial length,
$F_{width}(w)$ limits the road width toward their initial width,
$F_{angle}(n)$ limits the angle between adjacent road segment toward their initial values.

From raw data to suitable observation and parameters

We model the observations as either points or polygons associated with a confidence, a weight and a precision, and a classe of street object. theoretically, our use of polygon is generic enough so any observation type can be modelled (linestrings and points can be buffered), but we also use points for performance reasons. The weight allows some distinction for the same object type. For instance, an observation of sidewalk which has been observed three times should have an heavier weight than an observation of sidewalk observed once. The confidence, (spatial) precision and class of street object are outputs of the detection processes. We choose this basic representation of observations to be able to integrate observation coming from various sources, including low level sources.

We use street objects observations as surface. Kerb are also streets objects, yet, because they are so influential to determine road surface, and for performance reasons, we model them as points.

Because non linear least square is sensible to outliers and to control the quality and size of observations, raw data must be processed to create suitable observations. Figure 107 gives an overview of the various data sources we will use in this work and details in what follows.

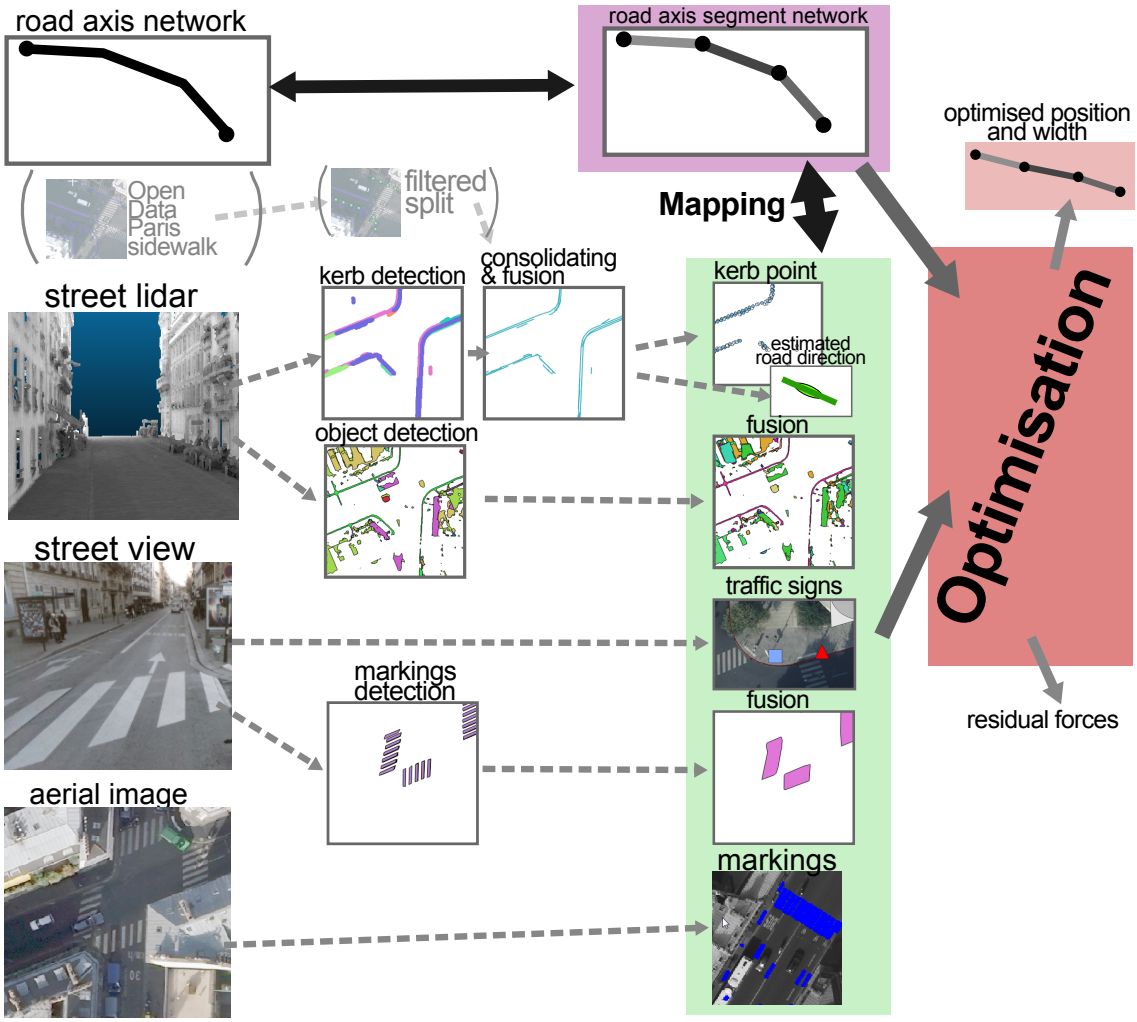


Figure 107: Data sources for optimisation. The input road axis network is refined into a road axis segment network. We use the detections from several methods. Those detections are processed to become useful observations, which are associated with road segments. The optimisation produces new road segment widths and positions.

Raw data for observation

Most of the raw data we use to extract observations come from a mobile mapping vehicle equipped with Lidar and camera (Paparoditis et al., 2012). It is essential to note that all sensing detections obtained by this vehicle sensors have a geospatial precision lim-

ited to the vehicle geo-positioning precision. Although areas with good GPS coverage have precision of under 10cm, the precision is also often in the 40cm range. These data are processed by various methods to extract information.

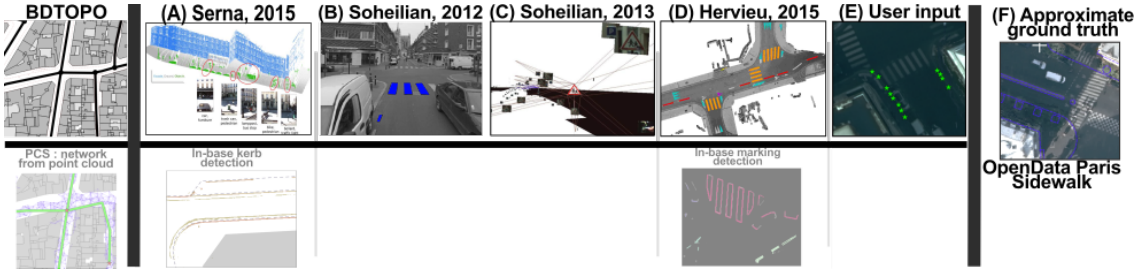


Figure 108: Overview of observations sources (and potential in-base alternatives).

(A) :OBJECT FROM LIDAR The main source of data available is the detection of kerbs and other street objects performed on street Lidar. We have access to results from Serna and Marcotegui, 2014 (in print), obtained on datasets from several hundreds of millions of points to 2 billions. (In the future we could also use in-base sidewalk detection from Cura, Perret, and Paparoditis, 2015b).

This method performs object detection (including kerb) on point clouds. For this, point clouds are rasterized (flattened), then, a gradient of height along with various morphological operation are fed to a random forest learning method. Connected components are then extracted, and processed with alpha shape to form polygons. We stress that raw results are extremely noisy, especially concerning labelling error for street objects. We performed extended filtering and consolidation to transform those detection into relatively reliable observations (We detail that in Section 5.3.3.3 on the following page).

(B) MARKING FROM STREET LEVEL IMAGES We have access to street markings extracted from street level images (Soheilian, Paparoditis, and Boldo, 2010). Those are polygons with basic semantic (type of marking based on width of line, and pedestrian crossing markings), associated with a confidence. Not all markings are detected (low recall), but erroneous markings are very uncommon (high precision). We performed basic consolidation to aggregate several detection of the same marking found when the vehicle passes several times at the same place.

(C) TRAFFIC SIGNS FROM STREET LEVEL IMAGES We also use results from Soheilian, Paparoditis, and Vallet, 2013 method, which detects traffic signs based on street level images through a multiview 3D reconstruction. Results are semantic points, with confidence and precision. Again the result contains very few false detections. Furthermore, it should be noted that in Paris, the signs are on the side-walk in the vast majority of cases. The result did not require any processing.

(D) MARKINGS FROM LIDAR AND AERIAL VIEW We have access to preliminary results from Hervieu, Soheilian, and Brédif, 2015. They use a sophisticated RJMCMC

framework to detect detailed markings with more diverse semantic from aerial images or rasterised Lidar. (In future work we could integrate marking detection from Cura, Perret, and Paparoditis, 2015b.)

(E) USER DEFINED KERB POINTS Interaction is essential (See chapter 4), therefore, we consider another observation raw data which is user defined points or polylines (converted to points (See Sec. 5.3.3.2)) representing kerbs. These geometries are associated with weight, so that they typically more important than observations from automated sensing. User defined kerb points are stored in a separate table, to allow easy logging and backup.

(F) KERB FROM GIS OPEN SOURCE DATA Lastly Open Data Paris [1] provides a sidewalk layer, which contains mixed data about sidewalks and urban features, including kerb polylines. Because this layer does not contain only kerbs, but also mixed information, using it besides visual ground truth is difficult. We filter this data to create an approximated quantitative ground truth, removing parts too small and using data semantics.

Beside creating an approximate ground truth, we can also create almost perfect kerb observations from these sidewalks (we convert again polyline to points).

from lines to points

Our optimisation method uses kerb observation in the form of points. In several cases, we have then to convert polylines to those points. (Using Open Data Paris sidewalk, using kerb detection from Lidar, using linestring user input). For this we split the lines so that no segment is bigger than l_1 (usually a few metres), then we assign the weight $\frac{SegmentLength}{l_1}$ to each segment node.

(A) Street object detections to observations

Curb detection from Lidar data is extremely noisy. Therefore, we had to resort to sophisticated filtering and consolidation. Part of the noise comes from the fact that the sensing vehicle passed several times in the same street, while its geo referencing systems encounter drift of up to 0.4m. We use a two step approach, with first consolidation of data and geometrical filtering, then contextual filtering. In both steps, we use data that could be reconstructed from sensing, therefore not loosing any genericity. We perform the processing using SQL queries and morphological operation with PostGIS.

Starting from noisy kerb points with confidence, the goal is to consolidate the points into polylines.

Starting from initial detected lines, we dilate (minkowsky sum with a disk) the lines with a radius of 0.4m (line spatial precision), then perform a boolean union of all obtained surfaces (unioned surface). For each new surface obtained, we transfer the confidence from initial points using weighted mean. We perform a straight skeleton (Aichholzer et al., 1996) on unioned surface, then filter the straight skeleton resulting lines to remove most of the minor radial segments. A simplification step reduces the geometrical complexity of lines (generalisation). Further contextual filtering must be performed,

1 http://opendata.paris.fr/

where (short) lines too close to buildings and crossing road axis are removed. (See Figure 109).

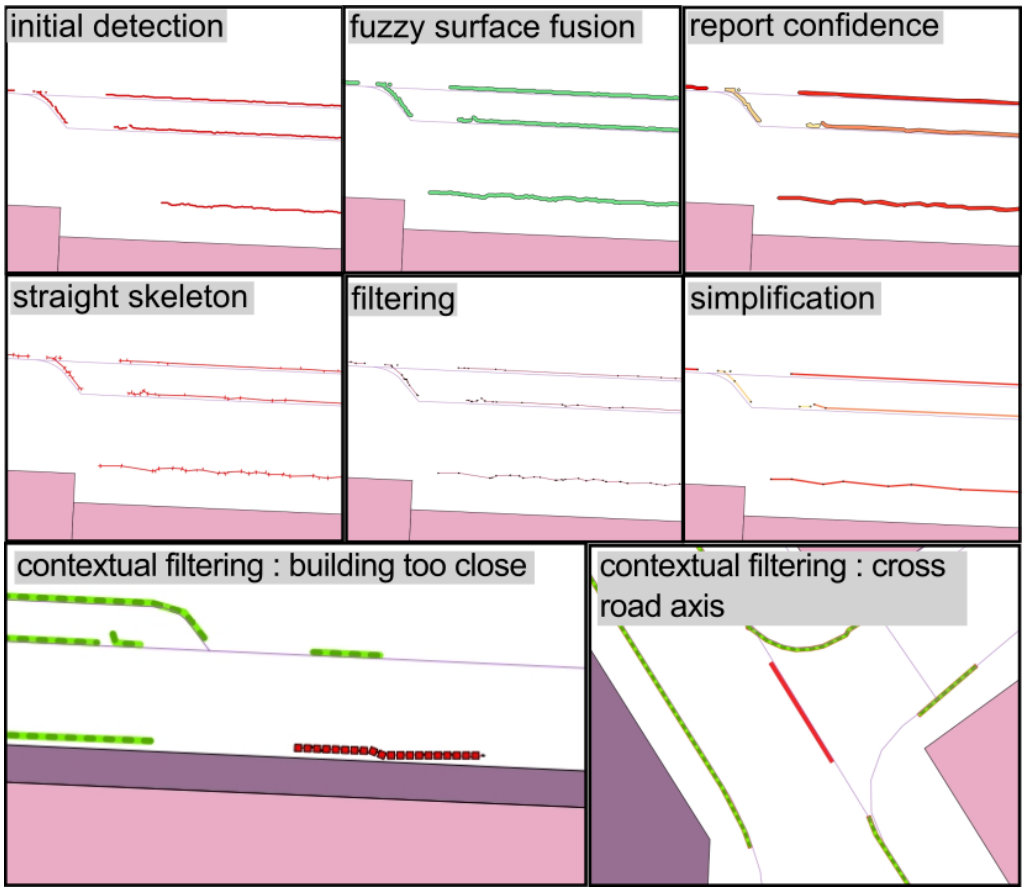


Figure 109: Input kerb detection is consolidated and filtered (both geometrically and with context) to produce suitable observations.

The cleaning process produces lines as output. We use these lines in two forms, as kerb points, and as kerb line segments. The first usage is to create weighted points (Sec. 5.3.3.2).

The second usage is to create segments of (almost) constant length l_2 that will be used to robustly determin a target slope of the road segment.

Please note that the example Figure 109 is a very favorable detection case. (See Figure 116 on page 169 for more common and challenging data.)

Road axis network

RAW DATA FOR ROAD AXIS NETWORK We use road axis geometry with approximate road width from IGN BDTopo. They consist of polylines with attributes. We create a PostGIS Topology from these axes, that is a graph of road axis polylines, with polylines being the edges of the graph, and the nodes being the intersections. Road axis precision is generally metric[2], approximate width precision is usually several meters.

2 http://professionnels.ign.fr/bdtopo

PREPARING NETWORK FOR OPTIMISATION The optimisation works on segments, therefore, we have to convert polylines into segments, while keeping its filiation to the road axis network topology. We perform these operations in base with custom SQL queries. Retaining the filiation allows to keep track of each road segment approximate width, as well as futur usage and analysis of results.

Linking observations to street segments

By design, each observation affects at most one street segment. Each observation must be attributed to a road segment, which we will call "match". We do not use "mapping" whihch is more accurate in database context to avoid any confusion with geographical mapping. Finding the optimal match between observations and road segments is a combinatorial problem. The chosen optimisation framework (robust non-linear least square) is not powerful enough to optimise simultaneously on matching and road model parameters.

However, we rely on a simple matching method (closest road surface), because streets tend to be less wide than city blocks, and because initial road axis and road width provide a rough road surface (precision of a few meters). Outside of intersections, these properties indicate that an observation can reasonably be matched to only one street, the other one being too far away to be considered anyway (See Fig. 117 on page 170).

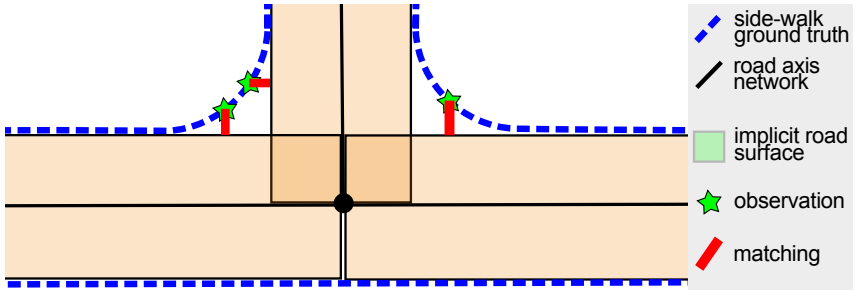


Figure 110: In intersections, kerb observations can not be used as the road model is not adapted. Using these observations would lead to incorrect results.

A notable exception is for observations near intersections, where our road model is clearly not adapted to curved kerb and curbstone inside the intersection (See Fig. 110). Please note that these observations are also amongst the noisiest due to limited curbstone height in Paris intersections. Therefore, we remove observations close to intersection surfaces. To determin intersection surface, we use Streetgen (See Chapter 3). (Still, those observations could be used to optimise the turning radius in StreetGen road model.)

This matching process (assigning observations to the closest road surface) is performed in base with carefully written queries. In theory, matching would require to consider $N_{observation} * N_{roadsegment}$ possibilities, which would become intractable at a city scale ($500k * 50k$ for Paris). However, we use database acceleration structures (Rectangle tree: GIST) so that only observation-segment pairs spatially close enough are considered. Processing is only a few seconds for about half an arrondissement (Paris is divided into 20 arrondissements), and a few minutes for the whole city of Paris.

Estimated road direction of road segment from observations

When matches between kerb observations and road segments have been computed, we use kerb observations to estimate a road direction (see Fig 111) for each road segment having observations. All observation segments are weighted by their length, then the direction of each road segment is computed and a weighted median performed on these directions. We choose the weighted median because it is not very sensitive to observation noise, while being fast to compute. The estimated road direction is then the weighted mean of the direction of observations having a direction close to the weighted median direction (by a threshold we experimentaly fix to 20 °).

For each kerb observation segment
split kerb observation linestrings to weighted kerb observation road segments (weight = length)
compute direction d_i for each weighted road segment $(s_i, w_i), i \in [1, N]$
compute directions weighted median $d_{wmedian} = weighted\_median(\{(d_i, w_i), i \in [1, N]\})$
estimated road direction = $weighted\_mean(\{(d_i, w_i)\}, i \in [1, N], |d_i - d_{wmedian}| < th)$
Where th is a threshold we experimentaly fix to 20 °.

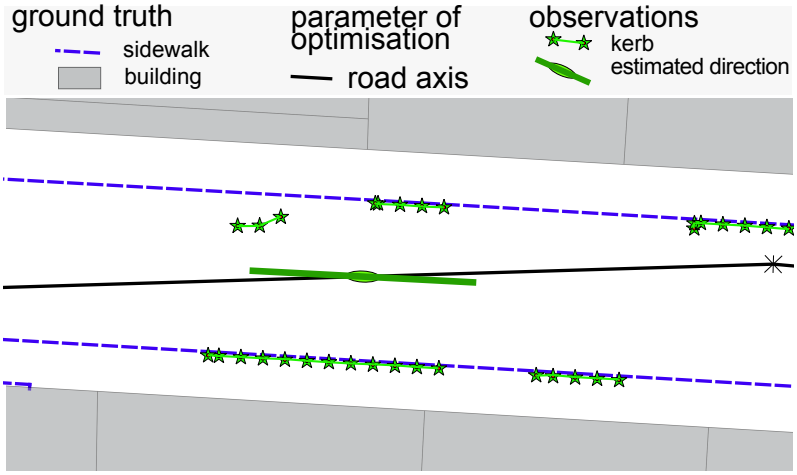


Figure 111: Estimated road axis segment slope based on weighted median of kerb observations. The estimated slope is coherent with the ground truth, and is very different from the road axis to be optimised.

Observation and regularisation forces

We model the optimisation problem as a mechanical problem, where each observation generate forces on the road model variables, and additional regularisation forces limits the variations of variables.

Processing of Section 5.3.3 produces a set of connected road segments with an approximate width (parameters to be optimized), and observations (weighted points or polygons, with confidence) matched to those road segments.

For our optimisation framework (Ceres-Solver), all forces are called constraints, with force values being called residuals, and force directions being called Jacobian (for each parameters).

Therefore each force defines a residual (whose squared value will be minimised), and a direction, that will be used to solve the optimisation problem.

Observation forces (F_o)

Observations generate forces over node position and road segment width (See Fig. 112).

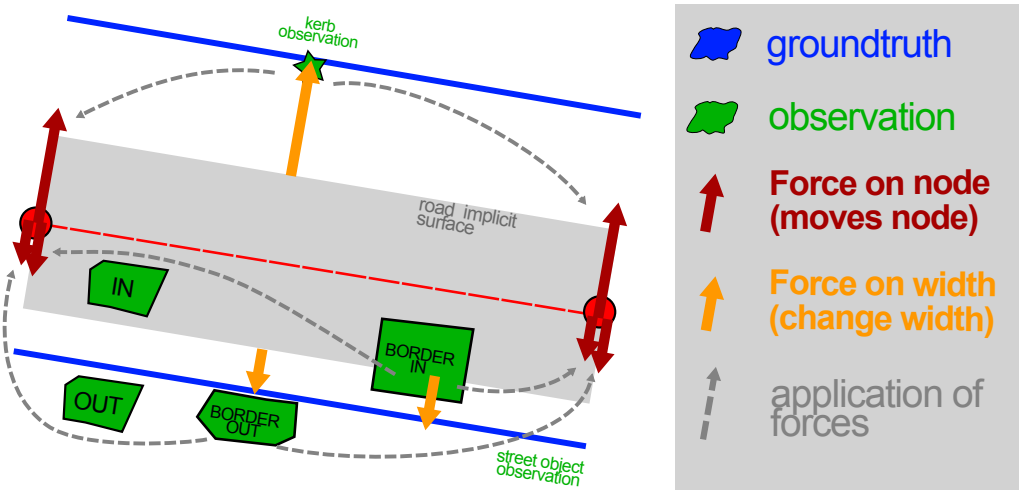


Figure 112: Example of forces induced by observations.

We define three forces from observations, as seen in equation 2 on page 155
$F_o(o, n, w) = F_{kerb}(o, n, w) + F_{object}(o, n, w) + F_{direction}(o, n)$

The idea is always that an observation generates forces that tend to change the road model parameters so the road model fits this observation.

Each force from observations exists in two versions; one affects node position, the other affects edge width. The residuals are identical, but the Jacobian changes.

FORCE FROM KERB POINTS (F_{kerb}) Each kerb point observation generates forces so the road surface border passes on this kerb point (See Fig. 112). Both road segment nodes and road width are affected. We use the same notations as in the open source implementation[3].) Given a road segment N_iN_j of width w, a kerb observation point Ob. The goal is to find the orthogonal distance between Ob and implicit road surface (i.e. implicit rectangle formed by road segment N_iN_j using road width w).
We compute $\overrightarrow{N_p}$ the normal of the plan P containing Ob, N_i, N_j with
$\overrightarrow{N_p} = \overrightarrow{ObN_i} \times \overrightarrow{N_iN_j}$.
Then $d = \frac{|\overrightarrow{N_p}|}{|\overrightarrow{N_iN_j}|} - \frac{w}{2}$.
d is the residual (whose squared value will be minimized).

3 https://github.com/Remi-C/Network_snapping/blob/master/using_ceres/Constraints.h#L409

For the version of the force affecting node positions, N_i and N_j changes happen in P, in an orthogonal direction to $\overrightarrow{N_iN_j}$, whose director vector is computed with $V_{ja} = \frac{\overrightarrow{N_iN_j}}{|\overrightarrow{N_iN_j}|} \times \frac{\overrightarrow{N_p}}{|\overrightarrow{N_p}|}$.

For the version of the force affecting width, the proposed width w_n is so $d = 0$, that is $\frac{w_n}{2} = \frac{|\overrightarrow{N_p}|}{|\overrightarrow{N_iN_j}|}$.

OBJECT OBSERVATION (F_{object}) We define another observation forces based on surface-like object observations (See Fig. 112 on the preceding page). Objects can be arbitrary polygons potentially having several inner holes.

The idea is similar to the kerb point force: the force is based on the distance between objects and implicit road surface border. As opposite to point to segment distance computation of kerb observations, the necessary segment to polygon distance for objects is not easily done. For this, we use GEOS [4] to find the distance between the implicit road surface (road segment and road width) and the object.

Each object class has an expected behaviour regarding road surface (IN, OUT, BORDER IN, BORDER OUT), that is used in the force. This distance also takes into account the expected distance between a class and the border of the road if the class is of type "BORDER". That is, objects of this class are expected to be at a given distance of the border. For instance, in Paris, a barrier is expected to be 0.2m from the border of the road (kerb).

Unlike points, a special case occurs when an object in neither entirely in or entirely out of the implicit road surface. In this case, we consider that the distance is proportional to the percentage of the object surface that is inside the implicit road surface, i.e. $\frac{Area(Object \cap RoadSurface)}{Area(Object)}$. This definition is generic enough to work well with any object we may use, and makes sense at the same time when considering the surface as the probabilistic location the object may be.

The version of the force that affects nodes has similar direction: orthogonal to edge axis in X,Y plan. The version of the force that affects width simply propose a new width so d=0 .

ROAD SEGMENT AZIMUTH FROM KERB POINTS ($F_{direction}$) For the two previous types of forces, the observations are considered individually. However this may lead to an ill conditioned problem. Strongly unbalanced observation density may stick the optimization in a local minimum. Intuitively, two points (at least) are necessary to determine a segment direction. To solve this problem, we use all observations affecting the road segment to determine a probable road segment direction d_o (See Sec. 5.3.3.6 on page 161).

The force is defined as follows. Given a probable target road segment direction estimated from kerb observation dir_o, the force rotates the road segment around its centre point so the road segment direction $dir_s = dir_o$.

4 https://trac.osgeo.org/geos/

Regularisation forces (F_r)

Observations generate forces over node position and road segment width (See Fig. 112). We define four regularisation forces, as seen in equation 2 on page 155
$F_r(n, w) = F_{position}(n) + F_{length}(n) + F_{width}(w) + F_{angle}(n)$

These regularisation forces keep the optimisation result close to the optimisation initialisation (See Fig. 113).

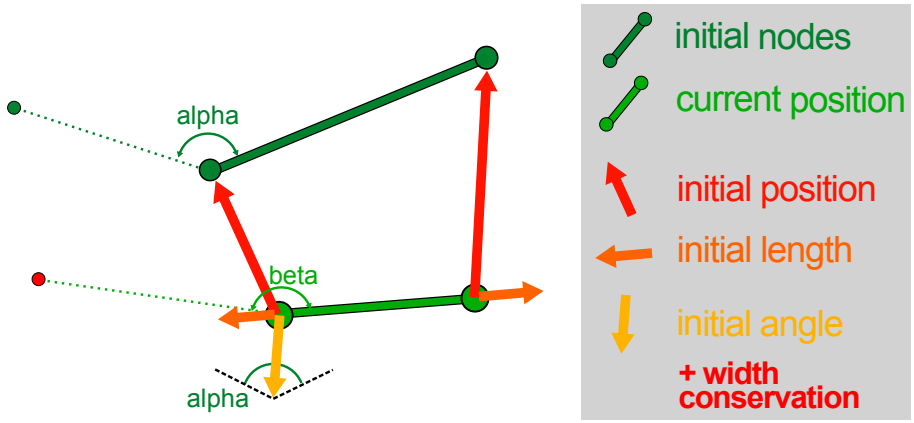


Figure 113: Regularisation forces that work to preserve initial values of parameters. The "initial position" force tends to preserve each node initial position. The "initial length" force tends to preserve each road segment length. The "initial angle" force tends to preserve initial angle between road segments. The "initial width" tends to preserve the initial width of each road segments.

In other words, the regularisation forces works to push the parameters towards their initial values, so as to limit changes and avoid large variation of parameters. Such regularisation are needed for three reasons. First, the initial road segment and road width (optimisation initialisation) are not far from the optimal solution (few meters). Second, road segments might not have any associated observations at all. As such, they could potentially be moved hundreds of meters. Third the optimisation does not perform validity checks on optimised road segments (for instance, we do not check if a new road segment position makes it intersect another segment, which would be forbidden in a topology).

The only parameters of the optimisation are road node position and road segment width, thus, in theory, only two regularisation forces would be necessary. Yet, regularisation forces are meant to preserve different properties of the initialisation. Lets take the example of a perfect road network, with perfect with and node positions values, but that is translated 1 metre North. Conserving the initial node position would prevent from translating the nodes 1 metre South to get the correct result. However, stating that angles between road segments shall be preserved allows the freedom to perform the translation, but still preserve the overall road network organisation.

We describe each of these regularisation forces ($F_{position}, F_{length}, F_{width}, F_{angle}$) in the following paragraphs.

DISTANCE TO INITIAL ROAD SEGMENT POSITION ($F_{position}$) Before the optimisation starts, the initial road segment node positions (X,Y,Z) are stored. Then the resisting

force is defined for each node as the euclidean distance between the initial position and its current position. The Force direction is $\overrightarrow{N_i oldN_i}$.

DISTANCE TO INITIAL ROAD SEGMENT LENGTH (F_{length}) This force is necessary to limit the deformation of the road segment network. Before the optimisation starts, the initial road segments length are stored. The resisting force is defined as trying to maintain the initial road segment length. Force intensity is defined by $F_i = Length_{initial} - Length_{current}$, the force direction is $\overrightarrow{N_i N_j}$, with direction depending on the sign of F_i.

DISTANCE TO INITIAL ROAD SEGMENT WIDTH (F_{width}) This force is particularly useful to limit large width changes when only a few noisy observations are present. The initial width is stored for each road segment. Then the resisting force is $|InitialWidth - CurrentWidth|$. This force only change width, and has no effects on node positions.

DISTANCE TO INITIAL PAIR OF ROAD SEGMENT ANGLE (F_{angle}) For each successive road segment node inside a road axis, we store the angle $N_i \widehat{N_j N_k}$. That is, when possible, we associate to a node N_j its initial angle ($\hat{A_j}$) with previous (N_i) and next (N_k) node.

The force direction is then the bisector of this this angle $\overrightarrow{B_j}$, which is computed with $\overrightarrow{B_j} = \frac{\overrightarrow{N_j N_i}}{2*|\overrightarrow{N_j N_i}|} + \frac{\overrightarrow{N_j N_k}}{2*|\overrightarrow{N_j N_k}|}$

Given the current positions of the 3 nodes, we look for the new position of N'_j so that $N_i \widehat{N'_j} N_k = \hat{B_j}$. Finding the actual distance d to the new position of the node N'_j along the bissect is complicated, as it require to solve a system so that $(\overrightarrow{N_j N_i} - d * \overrightarrow{B_j}) \times (\overrightarrow{N_j N_k} - d * \overrightarrow{B_j}) \cdot \overrightarrow{N_p} = tan(\hat{A_j}) * (\overrightarrow{N_j N_i} - d * \overrightarrow{B_j}) \cdot (\overrightarrow{N_j N_k} - d * \overrightarrow{B_j})$
where
$\overrightarrow{N_p} = \frac{\overrightarrow{N_j N_i} \times \overrightarrow{N_j N_k}}{|\overrightarrow{N_j N_i} \times \overrightarrow{N_j N_k}|}$.

Instead, we approximate computing of d by considering that $|\overrightarrow{N_j N_i}| = |\overrightarrow{N_j N_k}|$.

Optimisation

As seen in equation 2 on page 155, the optimisation problem is formed of optimisation variables (road segment node position, road segment width), and forces (from observations, for regularisation).

In this section we consider the optimisation process in itself, from how we influence it with several meta-parameters (weight, bounds, loss function), to which optimisation strategy we use, to how we can use the resulting road model to generate a street model with StreetGen (See Chapter 3 on page 88).

Meta parameters

First we use several level of weights to balance the level of confidence on different data and prior knowledge, second we bound the variation of the road model variable, then we weather the influence of outliers using loss functions.

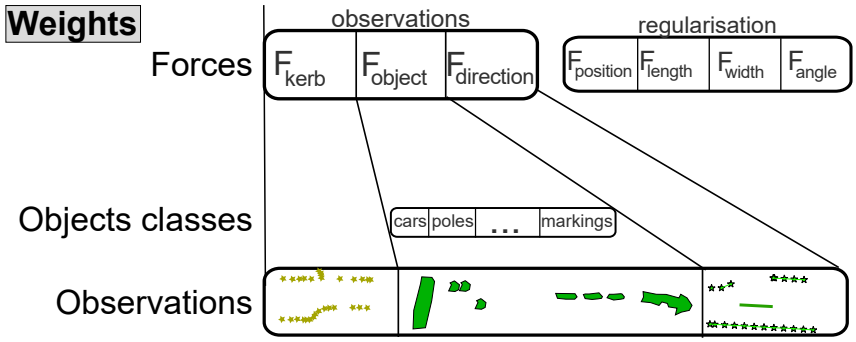


Figure 114: Several levels of weight allow to set the level of confidence in various data and prior knowledge.

WEIGHTS We use several levels of weights to adapt the optimisation process to data sources (see fig 114).

At the lower level, each observation is weighted. It controls how much faith the user has in this particular observation. If the observation is an object, each class of object is also weighted. This is necessary because some street objects are much harder to detect than others (for instance poles are easier to detect than trash cans), then this weight allows to express a preference between object types.

At the higher level, each type of force has a generic weight. This is crucial to balance between the confidence in observations and the confidence in the initial solution, or between forces. For instance, a user may know that the input road axis network is very precisely positioned, but with imprecise road width. This force-level weight is also what enables use of our methods in different scenarios. For instance, if observations are from noisy sensing, the weight of regularisation forces will be higher. On the opposite, if observations are from legacy GIS data, the regularisation force weight will be low, because we grant much more confidence to observations.

Choosing all these settings then depends on knowledge about input data and sensing data. We used an experimental approach to choose these settings.

BOUNDS Bound are used to limit the optimisation search space. For instance the road width is not realistically expected to vary more than by a dozen meters, neither a node to move more than a few meters. Bounds both limit the absolute range (e.g. width should be between 1 and 20 meters), and the relative range of parameters (e.g. width variation should be less than 10 meters).

LOSS FUNCTIONS Least square frameworks are very sensitive to outliers. Intuitively, an absurdly high value would have a very large squared value, which would in turn dominates the other regular values. Ceres-solver allows to use a classical solution to this problem: loss functions. Basically, instead of optimizing on x^2, we optimise on $f(x)^2$, where f is a function that acts like the square function at low scale, but is much flatter at high scale. In our case, we choose the "Soft L1" $f(s) = 2 * (\sqrt{1+s} - 1)$.

Meta strategy

We tested two strategies. The first is to optimise all parameters at the same time. The second strategy is to successively optimise for width and position, until no more im-

provement is reached. The first strategy is canonical and produces the best results, at the price of slightly more lengthy computation. The Second strategy is faster, but can be stuck in local minima and can produce worst results. As such, we consider the second strategy is not worth to be used, and favour the first strategy.

Generating streets from optimised road model

We use StreetGen to generate streets from the optimised road model (road axis segment network and road segment width), as depicted in Figure 115.

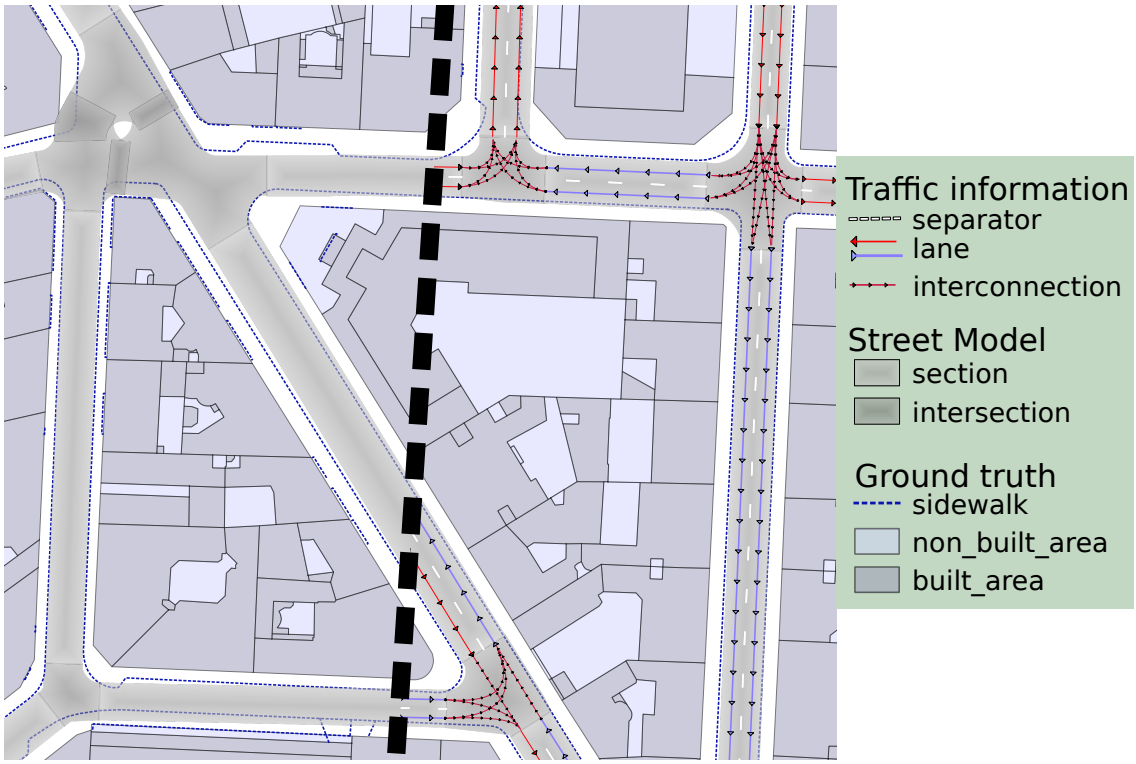


Figure 115: We use the optimised road model (kerb observation from sensing) as input to StreetGen to generate a complete street model. Note that turning radius have not been optimised.

REGROUPING ROAD SEGMENTS It is important to note that the StreetGen street model is based on polyline road axis associated with one width per polyline, while the optimisation road model is based on segment of road axis associated with one width per segment.

Before optimisation, each polyline is broken into segments.

A simple solution to use streetgen on the optimisation result would then be to consider that each road segment is in fact an individual road axis (a polyline composed of only one segment). This works in theory, but introduces many useless intersections. Indeed in StreetGen intersections are meant to deal with changes of road width and to deal with intersections of more than two road axis. In the optimised results, many road

segments of the same original polyline may have approximately the same width, and thus should be regrouped.

However, two factors complicate the regrouping. First, some segments may not have associated observations, and thus their width has not been optimised. Therefore, their width is insignificant and should be harmonized with the width of nearby segments from the same polyline that have observations (if any). Second, only successive segments should be regrouped.

We solve these issue in three steps. First, we work on segments having observations and so a significant width. For each polyline, we regroup the segments having approximately the same width using the DBSCAN method (Ester et al., 1996). The new width is the median width weighted by the number of observation of each segment. Second, we work on segments not having observations ("no obs"). We try to find other segments with a significant width in the same polyline. If any is found, the segment closest to the "no obs" (and with the most observations is two segments are at the same distance) shares its width with the "no obs". If none is found, the original width of "no obs" (un-optimised) is used.

In a third step, in each polyline, we regroup the successive segments with the same width.

TOPOLOGY CONSIDERATIONS Our optimisation method does not guarantee to preserve topology (no edge should intersect except in intersection node). Instead we use constraints so result is close to initial values. It suffices in most cases but not all (See Fig. 127 on page 178).

Indeed, a road segment could be moved in a way to produce an invalid topology. We detect these cases using a SQL query exploiting spatial index.

RESULTS

In this section, we first consider experiment settings and how to display results and forces (as visual control is as important as qualitative control for a road network). We present results for the pre-optimisation task, such as the transformation of raw data into observations, and the matching process between observations and road segments. Last, we present the optimisation results at different scales with different input data in part 5.4.4.

Resources

We use Ceres-solver[5] 1.10, the optimisation framework from Agarwal, Mierle, and Others, 2016. Our prototype implementation is available as free and open source software[6]. We use a Ubuntu 12.04 OS in a virtual box with dedicated 6 GB of RAM and 6 2.4 GHz Intel Xeon CPU threads. Timings are measured using the standard Cpp library.

5 http://ceres-solver.org/
6 https://github.com/Remi-C/Network_snapping

Results and Forces visualisation

Optimisation methods are notoriously difficult to control and parametrise, especially when input data contain outliers. Therefore, we consider essential to have a way to represent the forces, as well as the parameters, not only before the optimisation starts, but during the whole optimisation process (each iteration). Rather than create a new interface from scratch, we prefer to re-use a well known open source interface: QGIS. For each iteration of the optimiser, we compute forces and export them in Well Known Text (WKT), along with the iteration number and a (fake) timestamp, into a comma separated value text file. This text file is imported in QGIS and used with the TimeManager extension[7]. This extension creates a time slider to slide through the iteration of the optimiser, allowing to display both forces, residuals, and parameters.

From raw data to Observation

In Section 5.3.3.1 on page 156, we introduced pre-processing methods to filter and consolidate urban feature detections into suitable observations. The pre-process is especially necessary for kerb dection from Lidar data (Section 5.3.3.3).

Overall, the filtering and consolidating of kerb detection from Lidar data produce a much more tractable result than the initial one. The process is a few minutes long for about half a million detections (mostly due to the Straight Skeleton). The process can not remove all the errors, in particular because confidence measures are not giving much information. However, when observation density is sufficient, the optimisation seems to be robust enough to deal with the remaining noise. We compiled small examples of the remaining errors in Fig. 116.

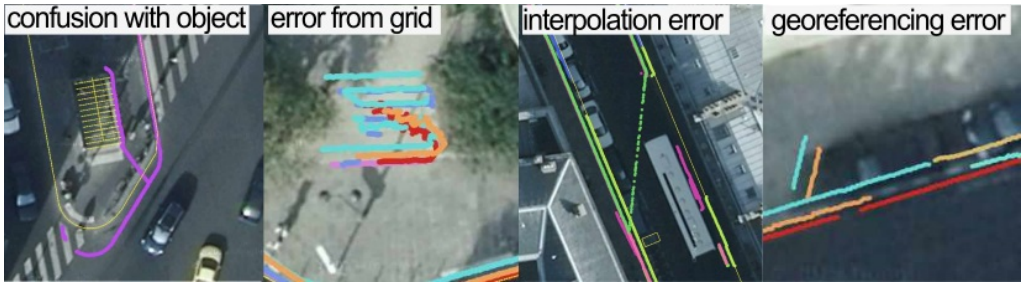


Figure 116: Input kerb detection is consolidated and filtered (both geometrically and with context) to produce more reliable observations.

Observations matching

Overall, the straightforward matching approach (closest implicit road segment surface) seems to work very well for kerb detection, and is fast thanks to our indexed approach (few seconds for sensing area). Object observation matching also seems to be correct (Fig. 117). We could find one case with incorrect matching, unsurprisingly in an intersection area (Fig. 117, right illustration, red circle).

7 https://plugins.qgis.org/plugins/timemanager

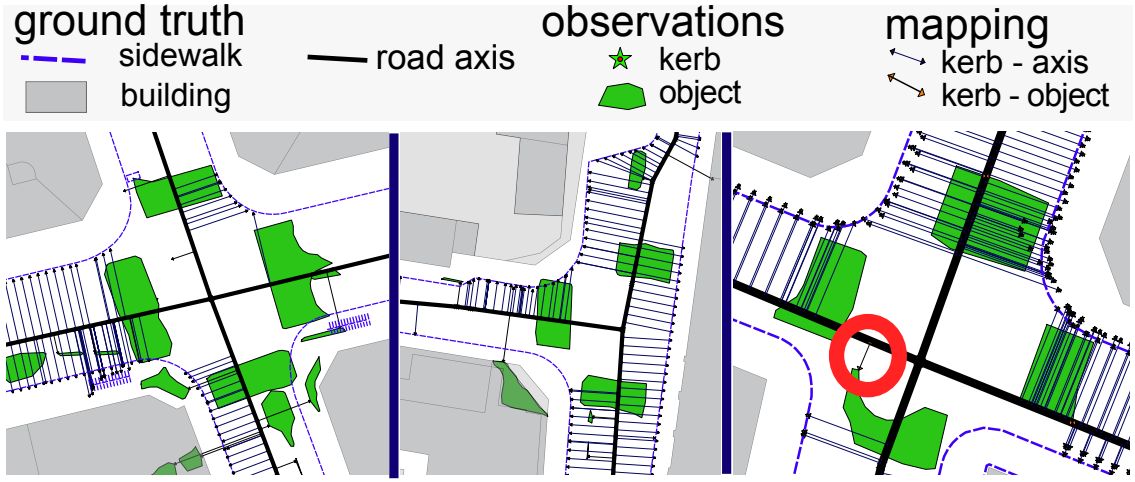


Figure 117: Observations are attributed to the closest implicit road surface, computed using road axis and road width. This simple matching works well, except inside intersections (Third figure, red circle).

Optimisation results

Result evaluation

Multiple factors introduce errors in the optimisation process, which complicates result evaluation (See Fig. 118).

- The ground truth is not perfect (it has not been updated since 2010) and it mixes side-walk with other urban features.
- The observations are noisy and sometime sparse.
- The input road axis network is not split enough. As our optimisation method does not change topology, this leads to an under-parametrised problem (or over constrained). A least square problem is by nature mathematically over-constrained (observations are redundant and contradictory). However, we refer to the fact that a human performing this optimisation would change the number of parameters by merging edge segment or splitting them.
- The chosen road model is not powerful enough to model all types of roads.

We design various strategies to limit these errors, allowing to analyse the influence of each source of error independently.

PERFECT OBSERVATION DERIVED FROM GROUND TRUTH Optimisation results strongly depend on observations quality (sparsity and noise). To remove this bias from the analysis, some optimisation are not performed on actual observations, but directly on ground truth side-walk points from Open Data Paris.

USER INPUT TO COMPLEMENT OBSERVATION SPARSITY Lack of observations has a big impact on optimisation. Therefore, we generate the best possible results with our noisy and sparse observations from sensing (Sec. 5.3.3.1). In a second step, we

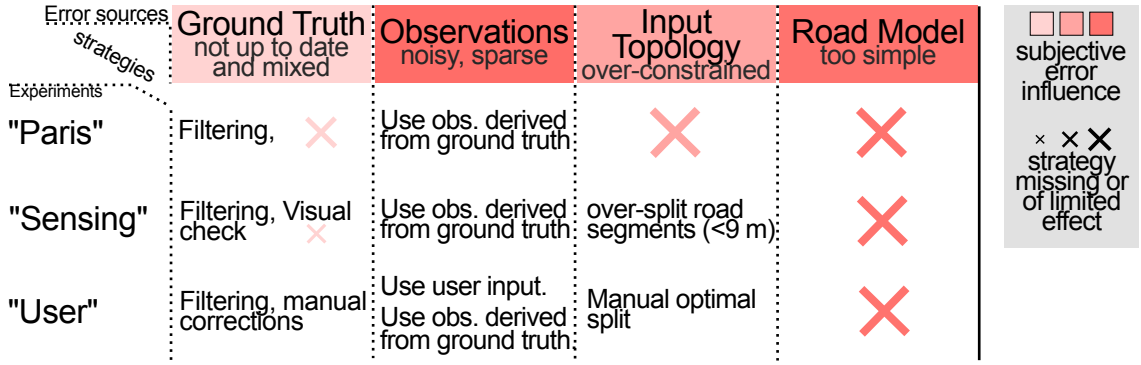


Figure 118: Multiple factors introduce errors in the optimisation process, which complicates result evaluation. We design various strategies to limit these errors, allowing to analyse the influence or each source of error independently.

introduce user inputs that will be used in the optimisation. Those user inputs are similar to StreetGen : user defined curbstone (kerb) points.

QUALITATIVE EVALUATION BY MEASURING RESIDUAL DISTANCE TO SIDEWALK We design an approximate qualitative measuring. The aim is to measure how much ground truth road surface is covered by the optimised implicit road surface formed by road axis and road with. There is no road surface ground truth for Paris, therefore, we use OpenDataParis side-walk border layer to get Paris sidewalks. We regularly place points on the ground truth kerb, avoiding the intersection area. The distance between kerb point and closest implicit road surface is then computed. The distance will be very small if implicit road surface fits well with the side-walk, and large otherwise. This measure can not be perfect because the side-walk layer also contains some street objects (not a perfect ground truth).

Experiment areas

CHOOSING EXPERIMENT AREAS Three different sized areas are chosen, each having a specific interest (see Fig. 119).

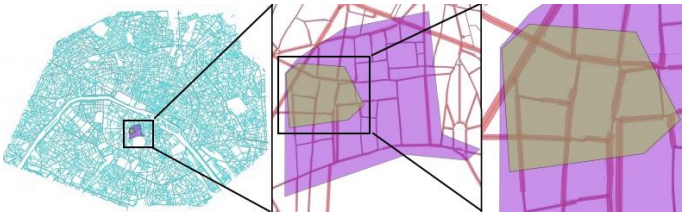


Figure 119: Three different experiment areas for various experiments, from the whole Paris to half an arrondissement to a few streets.

- The first area ("Paris") corresponds to the whole city of Paris. Sensing data is not available for all of Paris. Therefore, we do not use observations from sensing data, but instead, we create observations from ground truth (Open Data Paris Sidewalk).

Area	# edges	# nodes	# Groundtruth Observation	# Kerb Obs.	# Car Obs.	# Markings Obs.	# all objects Obs.	# User Input
Paris	38.8 k	45 k	522 k					
Sensing	186	215	2343	11.7 k	390	1333	5523	
User	62	64	475	2628	145	136	2011	12 places

Table 7: Facts for ground truth, parameters and observation for the three experiment areas.

This area is used to demonstrate scalability, analyse the use of constraints, and check the optimisation robustness in extremely complex parts of the road axis network.

- The second area ("Sensing") is the whole area where sensing data is available. Road types and observations are very diverse. This area is used to determine how useful a fully automatic optimisation process would be, as well as evaluate computing time for this amount of object observation.
- The third area ("User") is a small area with sensing data available. We purposefully chose the most challenging area regarding road surface complexity. Due to its small size, road axes can be manually split into the relevant number of road segments, thus eliminating the over-constrained bias of the evaluation. A thorough visual control is also possible. Last, the moderate size allows some experiments on the usefulness of user inputs.

NUMERICAL FACTS ABOUT EXPERIMENT AREAS Table ?? shows an overview of experiment areas size and content. Optimisation computing time is between a few minutes ("Paris"), a few seconds ("sensing") and less than a second ("user") when no object observation is used. Using object observations greatly slows the optimisation process as it relies on the GEOS library to perform geometrical computing at a great cost. As such, optimising time is a few minutes for "sensing" and a dozen of seconds for "user".

Results on "Paris" area

The first subjective result is that optimisation is very successful, the fitting of the model is greatly increased. The qualitative result (See Table 8) confirms this (median distance to ground truth diminished by a factor of ten, from 1.547m to 0.104m). At the same time, because optimisation is performed on ground truth observations, the fact that the result is not perfect shows that the model is not expressive enough for all types of roads, and that it is over-constrained.
However, these two bias (model too simple, over-constrained) do not prevent the results to be very close to ground truth.

The second result on the "Paris" area is that optimisation scales very nicely. The area represents about 170k parameters, and around half a million observations, plus regularisation constraints. The entire optimisation is done in 3 to 4 minutes, with about 30 seconds spent on reading data and constructing the problem (in a Ceres-solver meaning).

The third result is that optimisation is robust, even when the input road network is extremely complex (see Fig. 120).

We also looked at the impact of the regularisation constraints that limit changes (See Section 5.3.4.2). Clearly, constraints can degrade the result, leading to less correctly fitted

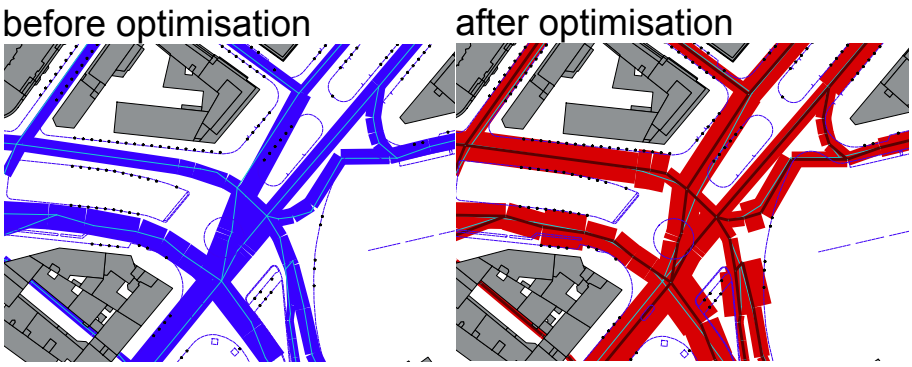


Figure 120: Even in very complex situations, where observations are too sparse and the simple road model is not sufficient, results are stable (no constraints used).

roads (See Fig 121). This phenomenon is particularly showing when looking at distance between ground-truth side-walk and road surface envelope (Table 8), especially looking at histograms (122). We stress that constraints can stille be introduced, but with an influence reduced at will using force weight. That way, the constraints weight can be adapted to the trust the user has in the observation and road model parameters initial values.

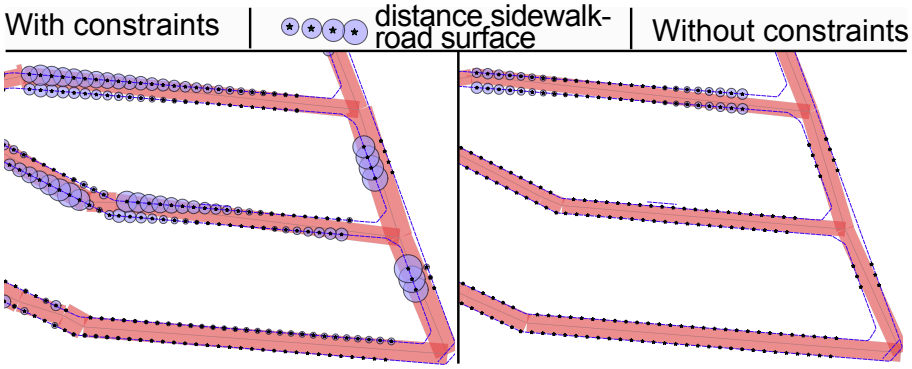


Figure 121: Constraints are necessary to stay close to the initial solution. However, they can significantly degrade the results when there are enough quality observations. Circles are proportional to the distance between ground truth and result. Note that constraint influence can be modulated with weights.

Table 8: For whole Paris, distance to side-walk before and after optimisation using or not regularisation forces.

Type	Mean (m)	Median (m)	Std dev (m)
Initial (no optimisation)	1.797	1.547	1.357
Observation from ground truth (ODParis Sidewalk)	0.392	0.242	0.479
Observation from ground truth, no regularisation	0.316	0.104	0.638

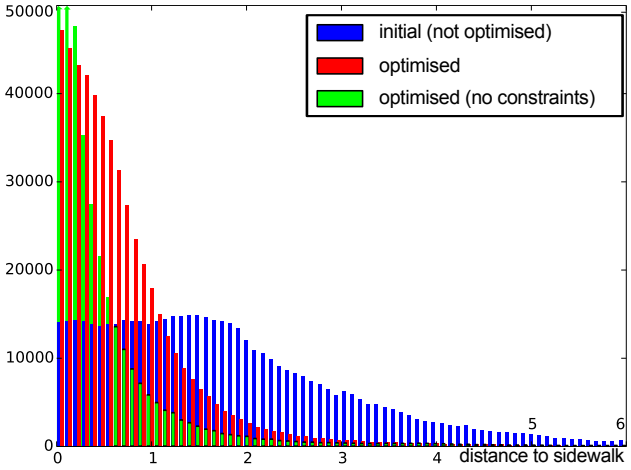


Figure 122: Histogram of absolute distance to ground truth side-walk for "Paris" (optimisation using ground truth as observations).

Results on "sensed" area.

The second area of experiment is the place where sensing data are available (See Sec. 5.4.4.2 on page 171).

Whatever the observation type used, the fitted model is always better than the initial road modelling, although using only object-observations improves only marginally the fitting.

Results using kerb observations are subjectively good when enough observations are available. Quantitative results (Table 9) confirm this impression.

However, there is a large difference between results obtained with observations from the ground truth (median = 0.12 m) and results obtained with kerb observations from sensing (median = 0.61 m). The visualisation of observations and distance between ground truth side-walk and road model immediately informs about the problem (See Fig. 123, bottom right illustration). Errors come mostly from sparsity of kerb observations (most often no observation, and in a few cases only one side of the road). Indeed, finding the correct road segment and road width is not possible if observations are only available on one side of the road, or not available!

If the whole optimisation process were to not use any user inputs, using sensing data clearly improves fitting (median distance to ground truth is optimised from 1.51m to 0.61m), but it still remains far from reachable results with more extensive observation coverage.

Figure 123 gives an overview of results on whole sensed area.

Results on "user" area

The last experiment area is a small part of the "sensing" area. We manually ensure that the road segment network is sufficiently divided. The initial road model (Fig. 124, top right, viewed with StreetGen) is quite far from the ground truth (top left), especially concerning road segments width. Observation matching (bottom left and right) shows the sparsity of observations (especially kerb) and noise (especially objects).

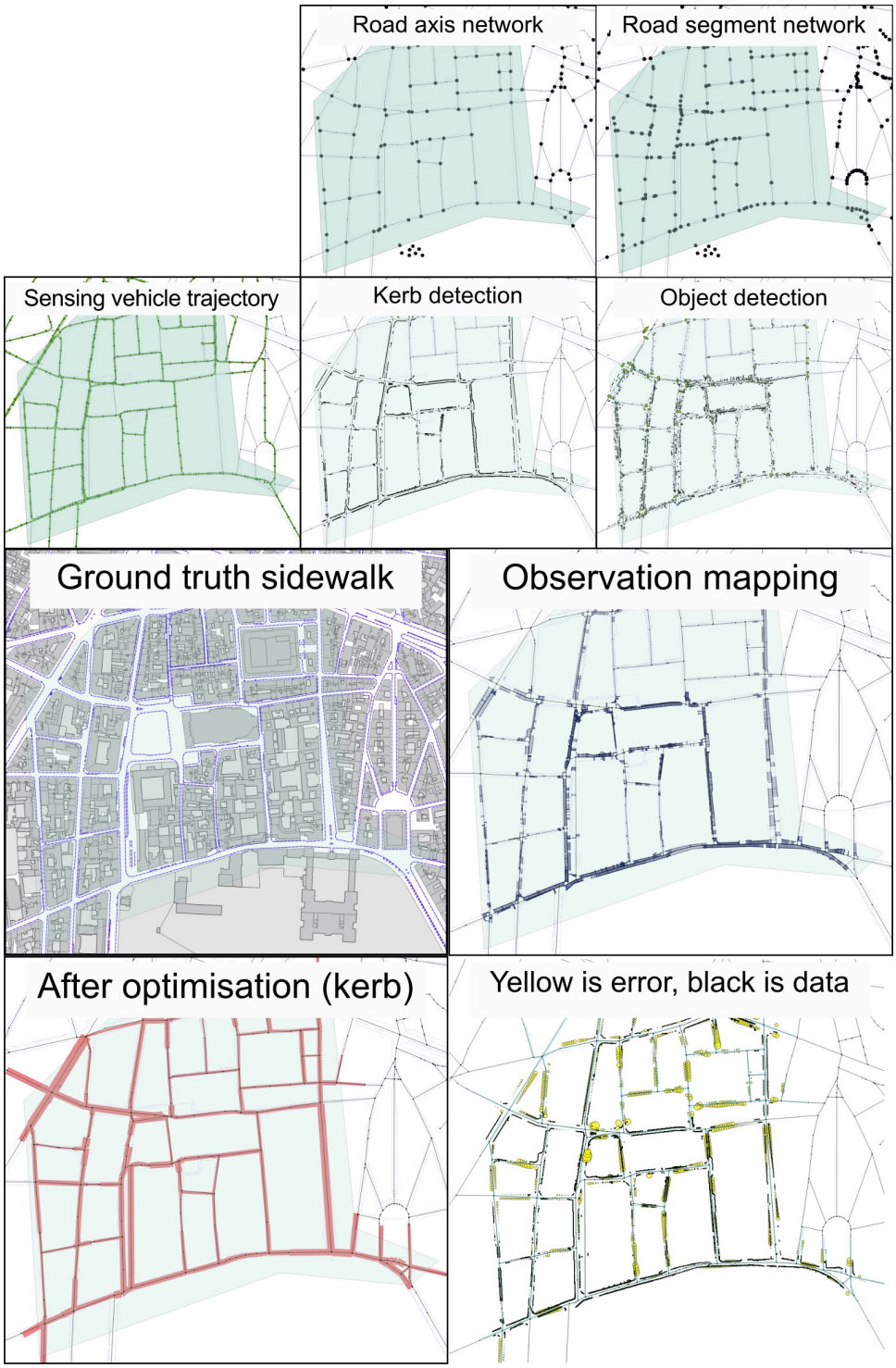


Figure 123: Optimisation parameters are the street axis network and width attribute, which get split into a road segment network. A street sensing vehicle detects kerbs and objects, which are processed into observations, and matched to street segments. Non-linear least square optimisation fits the parameters to these observations.

Optimising using kerb observations produces a convincing fitting where observations are available (see Fig. 125, left, and Table 10). Because bad fitting happens when obser-

Type	Mean (m)	Median (m)	Std dev (m)
Initial (no optimisation)	**1.856**	**1.507**	**1.534**
Observations from ground truth (ODParis Sidewalk)	0.481	0.356	0.469
Observations from ground truth, no regularisation	0.281	0.118	0.442
Observations from ground truth, no regularisation, split road seg >10 m	**0.117**	**0.014**	**0.306**
Using Kerb observations	0.825	0.608	0.783
Using Object observations	1.554	1.237	1.201

Table 9: For the entire sensing area, distance to side-walk before and after optimisation with diverse observations.

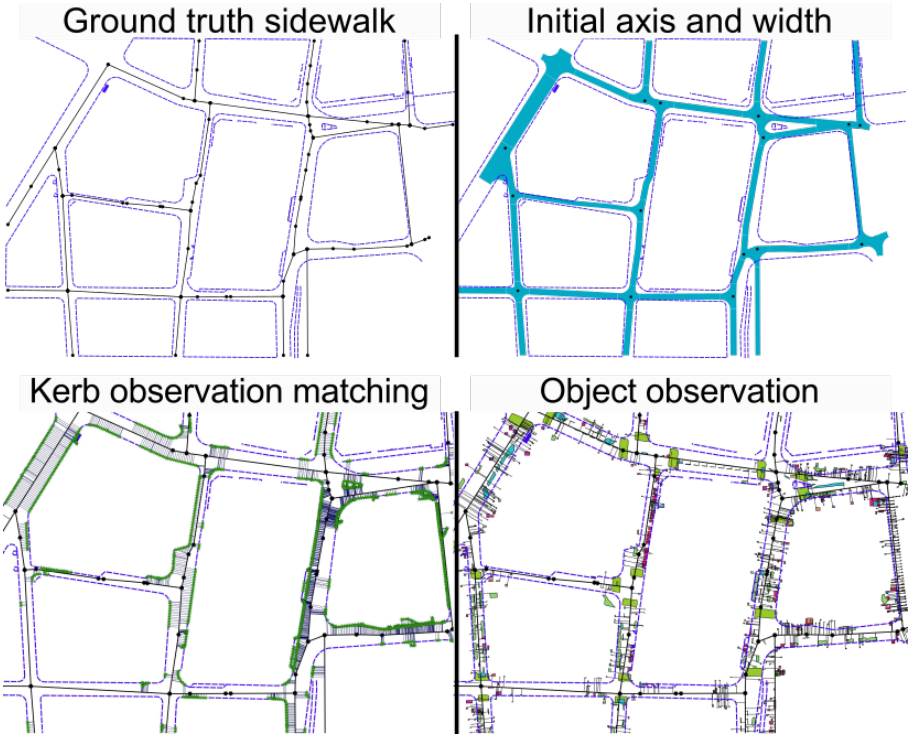


Figure 124: Input data (road axis network + width with matched observations from sensing) for optimisation in the "user" experiment area.

vations are missing, a user can input manually relatively few kerb points (middle). The resulting fitting after optimisation (Fig. 125, right), is extremely close to the ground truth. However, the distance to ground truth (Table 9, row 4) (median = 0.36 m) is still higher than the median distance reached for all of Paris with "perfect" observations. This resulting distance is then a sign of the limit of the expressiveness of the proposed road modelling (road segment + width). For instance, when the road width increases only on one side of the road. Indeed, the chosen "user" area contains many intersections, and problematic situations, like road with non constant width, and complex pedestrian crossings with pedestrian islands in the middle of the roadway.

We further test the missing data hypothesis with an alternate kerb observation data set, with more coverage, but also much more noise. As expected, the results are better (Table 10), the additional noisiness is mostly dealt with by the loss function.

We also experiment using object observations (Fig. 126). Object observations are less noisy than in other places, and so they can be used for optimisation on their own to

obtain notably better fitting (Table 10), with median distance going from initial 1.8m to 0.96m. However, results are still far from those obtained with kerb observations.

To complete this, we use marking detection from Hervieu, Soheilian, and Brédif, 2015, which are especially complete and with a limited noise (advanced markings). After fusion and optimisation, the result is significantly improved compared to other object observations (median distance of 0.78m).

(See Discussion (Section 5.5.7.1) for analyse of other results of table 10).

Type	Mean (m)	Median (m)	Std dev (m)
Initial (no optimisation)	**2.044**	**1.8**	**0.937**
Ground truth (ODParis Sidewalk)	0.394	0.237	0.448
Ground truth (no constraint)	**0.22**	**0.084**	**0.404**
Using Kerb observations	0.709	0.455	0.724
Using Kerb observations and User input	0.457	0.34	0.413
Using Kerb observation (alternate d.)	0.496	0.39	0.413
Using Object observations	1.344	0.963	1.311
Using Object observations (only markings)	1.53	1.169	1.272
Using kerb and Object (cars,markings,signs)	0.67	0.451	0.696
Using advanced markings (only few streets)	1.009	0.782	0.944

Table 10: For a small part of sensing area, distance to side-walk before and after optimisation using diverse observations.

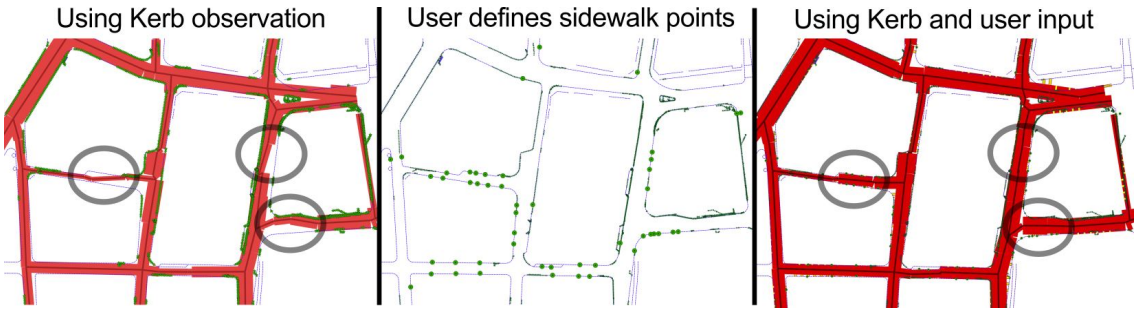


Figure 125: Optimisation results based on kerb observations for "user" experiment area.

Generating streets from optimised road model

In Section 5.3.5.3 on page 167 we introduced how the optimised road network is used to generate a street network using StreetGen (Chapter 3). We test the method on two optimisation results: first the road modelling obtained on the "Sensing" experiment area using kerb observations, then the road modelling obtained on the "Paris" experiment area using the observations extracted from ground truth side-walk, with a road axis network split so no road segment is longer than 30 m.

The process to regroup road segments is quite fast, with a few seconds for "Sensing" and a minute for "Paris". See figure 115 on page 167 for illustration on result. Produced roads and intersection surfaces are available[8].

8 https://github.com/Remi-C/Network_snapping/tree/master/data/street_gen_after_optim

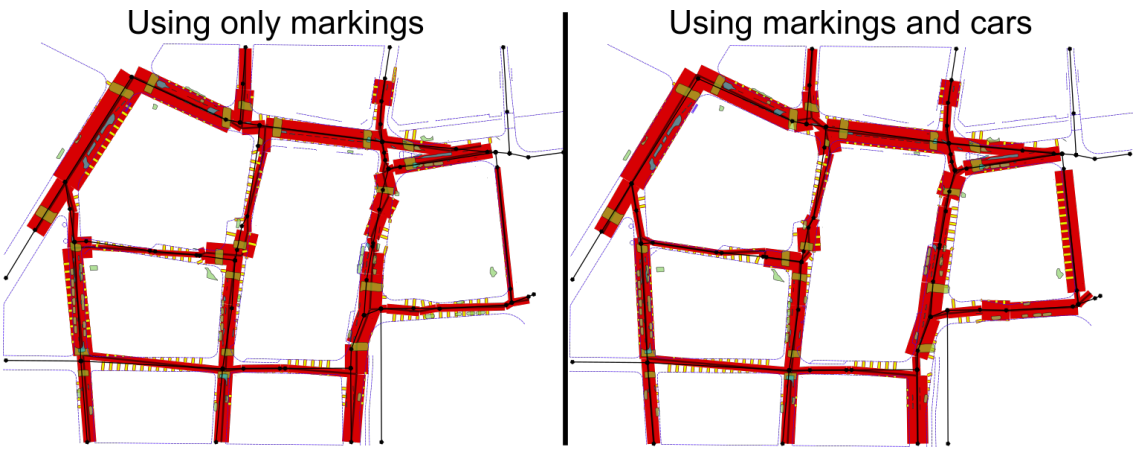


Figure 126: Optimisation results based on object observation for the "user" experiment area.

For "Paris" experiment area, the initial road axis network contains 19531 polylines, which are then split into 74325 road segments. After optimisation, those segments are regrouped into 37262 polylines.

Optimisation does not guarantee to preserve topology (See Fig. 127). We detect these errors (74) and display them, so an user can fix them.

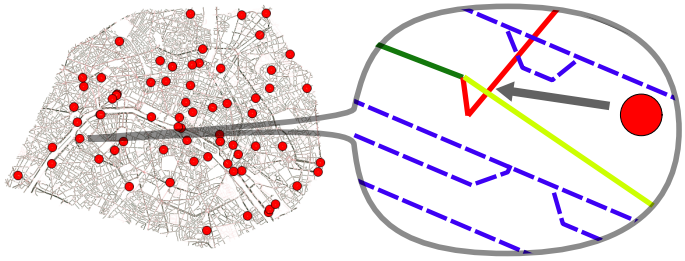


Figure 127: After the optimisation process, the road axis network contains a few topological errors despite the use of constraint forces. These 74 errors for the whole "Paris" experiment area can be corrected by a user in about 10 minutes.

DISCUSSIONS

This section discusses limitations and perspectives for key elements of our method and its results. First, we discuss the choice of the road model, optimisation framework and forces choices. Then we discuss how raw data are processed into observations, and how these observations are matched to road segments. We then discuss the optimisation process and how to visualise its output. Last, we discuss optimisation results.

Modelling the problem

Our road modelling (road axis segment + width) suits well a city like Paris that has yet very complex roadway. Our road model can not take into account road with non

constant width, asymmetric road width change, pedestrian islands in the middle of a roadway, etc.
In fact the current optimisation method could be extended in a minor way by introducing a road segment type with a linearly varying road width (possibly with variation different on each side). Such an extension would even be sufficient to model asymmetric road width change in most situations. This change would greatly enhance the capabilities of the road model.

Another perspective would be to augment the road model with some "free-form" road sections that would be much less constrainted that regular road sections (for instance, any polygon). This could greatly improve the expressiveness of the road model, at the price of it being partially less abstract.

The choice made toward robust non linear least square optimisation proves to be capable of fitting the chosen model to observations, and is reasonably robust to outliers in observations. Because it is fast, it is a good candidate to be integrated into a more powerful framework such as Reversible Jump Markov Chain Monte Carlo (RJMCMC, Green, 1995) which can handle changes on numbers of parameters, extension of road model into different type of road segment, and possibly remove outliers.

Modelling observation effect as forces

We consider the optimisation problem as a mechanical problem where observations produce forces on road segments and road width.

Observation forces

KERB POINTS Forces induced by kerb points are extremely fast to compute thanks to Eigen[9] (Guennebaud, Jacob, and others, 2010) library. These forces have a drawback: distance between observation and road segment is computed for an infinite road segment. It could be an improvement to actually check that the observation projection is inside the road segment, which would amount to disable some observations when road segment changes a lot.

STREET OBJECTS Forces induced by road objects have a lot of room for improvement. We use GEOS[10] to compute forces based on object observations, which greatly increases computing time (by a factor 10 to 100). We chose a sophisticated distance, based on the actual shared area. It involves computing distance between road segments and complex surfaces, and computing intersections between road segment surface and observation surfaces. The reason behind was to exploit exactly the data, without introducing approximation. However, there is no point to use such a precise measure when object detections and observations are so noisy! Moreover, observations are systematically converted to surfaces. This is inefficient: an object observed as a point (a pole for instance) could use a point to segment distance. Considering the level of precision of the observations, GEOS could be abandoned to perform surface to segment distance and intersection area. Instead, all object observations could be modelised as points or rectangles, and distances and intersection could be directly computed with Eigen.

9 http://eigen.tuxfamily.org
10 https://trac.osgeo.org/geos/

ROAD SEGMENT AZIMUTH FROM KERB POINTS Target road segment slope has proved to be successful at accelerating computing and obtaining better results. It may be due to the robust (although crude) way it is estimated (weighted median). More advanced methods such as RANSAC could make it more robust. Moreover, this force is sometimes meaningless for very short road segments that serve as transition in places where the road model is not sufficient compared to the road complexity. Such cases could be detected automatically, and slope force cancelled. User inputs are not included to compute this force, which could also be a significant improvement.

Regularisation forces

All regularisation forces suffer from a drawback: they are hard to set, because their value is a constant, whereas observation forces strength depends on the number of observation. Regularisation forces could be normalised regarding the number of observations, yet their current behaviour can be seen as desirable (a road with many observations will have low regularisation, and with few observation strong regularisation).

NEW ROAD WIDTH PROPAGATION FORCE We could improve results in case observations are missing by regularising the width given the neighbour segments width. Let us take the example of a road axis divided in three road segments, with good observations on the first and last segments, but no observations for the middle segment. First and last segments are moved and their widths changed so they fit the observations. Indirectly, the middle segment is also moved as its end nodes are shared with other segments. However, there is no such mechanism for width sharing. We could easily add a force that pushes toward having the same width as neighbour segments. This force would greatly increase results in place where the road axis network is over segmented.

Similarly, we could add a force limitating the difference of width between two successive road segments.

DISTANCE TO INITIAL ROAD SEGMENT NODE POSITION We consider this force to be very important for scenarios where only a small part of the road network is to be modified. Indeed, it would be critical for user interaction that a change made by a user at a place (South of Paris for instance) does not produce changes far away from this place (North of Paris for instance). This force is a way to achieve that. Let us take the example where the user wants to edit one road axis. This force can be gradually stronger for edges as they are inside 1-neighbourhood, 2-neighbourhood, etc.

In a similar scenario, this force is useful for pending edges that are directly connected to optimised edges but have no observations.

DISTANCE TO INITIAL ROAD SEGMENT LENGTH This force could also be modulated, as it is counter-productive for small segments that serve as transition whent he road model is not powerful enough.

DISTANCE TO INITIAL ROAD SEGMENT WIDTH This force works well to prevent isolated noisy observation to wrongly radically change road width. It proved to be particulary useful when dealing with noisy objects, as it prevents the width to increase out without sound reasons.

DISTANCE TO INITIAL PAIR OF ROAD SEGMENT ANGLE We performed an approximation to compute this force due to lack of time. Yet this approximation may be ill suited when changes of angles are large, possibly leading this force to be inefficient or even counter productive. We possibly introduce an error in optimisation. The exact analytical change could be computed, which may lead to faster convergence, and maybe even better results.

From raw data to observation

STREET LIDAR FOR KERB OBSERVATIONS The main limitation of kerb observations is not the noise but the sparsity. It has a much stronger effect on optimisation outcome than outliers. Some streets have very few kerb detections, sometime on only one side of the street. The essence of the problem is that side-walks are often masked by parked cars or street objects. This is due to the used Lidar device (Riegl), which samples points in a plan orthogonal to the vehicle heading. That way, a spot of side-walk may only be seen from one angle, which would then be easily occluded.

We considered the interest of using another Velodyn Lidar as a complementary Lidar. The Velodyn lidar is rotating, and thus can sense points from many different angles when the vehicle closes in and out (See Fig. 128). Figure 128 clearly shows that Velodyn greatly increases side-walk and kerb coverage. If the distance is sufficient, kerbs can even be sensed with ray passing under the cars.

However, the Velodyn point cloud is also noisier and much larger. With increased data volumes and a need to fusion points, Velodyn raises practical challenges. The main one is that as opposite to Riegl, point clouds can not be processed linearly, file by file. Indeed a portion of sidewalk may be seen by the Velodyn Lidar in many files, not necessary temporally close to each other. This problem is hard to solve with a file based solution, but would be quite easy to deal with the Point Cloud Server (See Chapter 2).

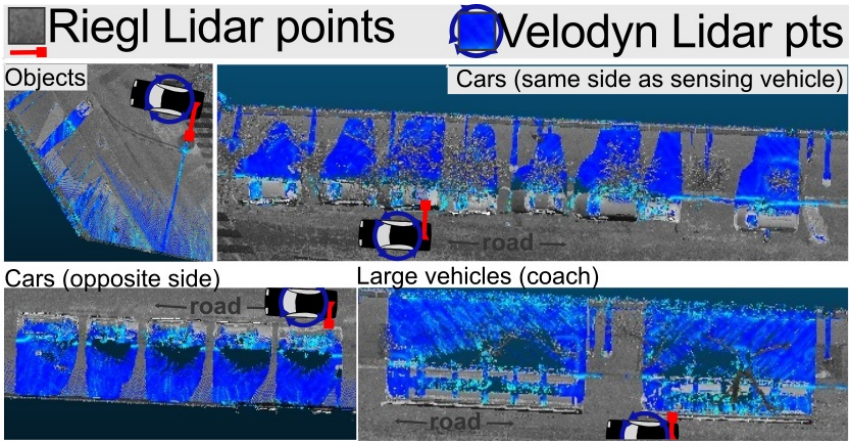


Figure 128: Point cloud sensing with static Lidar (Riegl) is occluded by cars and street objects, which hides side-walks. Adding a rotating Lidar (Velodyn) greatly reduces occlusion because the same spot can be sensed from different angles.

STREET LIDAR FOR OBJECT OBSERVATIONS Objects produced by Serna and Marcotegui, 2014 (in print) are most often too noisy to be used. In some cases, carefully

filtered car observations can be used for rough road fitting (such as in "user" experiment area).

MARKINGS We mainly use markings detected from stereo-vision (Soheilian et al., 2013), with a low semantic content and only the main markings detected (pedestrian crossing and center lines). However, a more powerful new method (Hervieu, Soheilian, and Brédif, 2015) has been shown to detect more diverse markings, with the potential to detect parking place markings. Those markings are important because they are close to the road border, and thus give a good estimate of road width. Indeed, a central line marking is currently not very useful for the optimisation, as there are very few chances that this markings would be outside of the initial road surface.

Observation matching

WRONG MATCHES Intersections are ambiguous place for matching, partially because our model (road segment + width) is blatantly wrong in those places. For kerb observations we simply exclude observation within intersection area. Yet we can not proceed similarly for object observation, as some objects are most frequently found in intersection area, such as pedestrian crossings and cars.

The problem is that a large object may be at 0 distance of two road segment (i.e. intersection both). A a supplementary criteria to distance to implicit road surface could be added to disambiguate in this cases. The additional criteria could be a robust shared area measure, where the implicit road surface sharing the most area with the object to be mapped would be the one the object would be mapped with. Another solution would be to add object knowledge, such as orientation, and favor matching object with road having a similar orientation. Although it would be an easy change, we feel it potentially breaks the genericity of the input we use.

DYNAMIC MATCHING The chosen optimisation solution is sensitive to outliers, be it real incorrect observations, or observations that are incorrectly matched. We stress that even if the matching is correct before optimisation, it may become wrong during optimisation when the edges are moved. We detected several cases of such matching turning wrong. A solution could be to perform dynamic matching. The first way would be within the optimisation, for each iteration, observation generating large forces are re-mapped or deactivated. The second more discrete way would be to perform several successive optimisations, each time re-matching what looks like outliers, possibly removing them. In this approach residual distribution of forces can be statistically analysed to find outliers, but it remains an adhoc process.

OPTIMAL MATCHING Another perspective would be to include the matching in a more powerful optimisation process. In this case, the system would optimize the parameter values, the number of parameters, as well as the matching between observation and parameters (including removal of outlying observations). This solution would be theoretically sound and could rely on good initialisation, but the design and more importantly balance between energy terms might be difficult, and the computing time potentially prohibitive.

Optimisation

Several settings can be changed for optimising. Main settings are weight of each type of forces, and scale factor of loss function. In total, it amounts to less than 10 program-settings. We choose those settings with an empiric trial and error methodology, which is less than satisfactory.

A ground truth road segment network and width could be manually entered for a small area, and then those program-settings could be found with a meta optimisation by measuring the distance to user-defined road model, either with stochastic method or with a similar non linear least square (although gradient would have to be obtained numerically and not analytically).

We consider this to be essential to analyse the full potential of our optimisation method, as we could currently be using non optimal program-settings.

Ideally, we would not only find the best parameters for road axis segment position and width, but also be able to introduce new segments or delete unneeded ones. Similarly, parameters are changed to fit observations, yet some of this observations may be erroneous, and should be ignored. This would require an optimisation method that not only find the optimal values for parameters, but also find the optimal number of parameters, along with the optimal set of observations! Methods with this kind of capabilities are pretty limited to direct approach(brute force), more or less clever heuristics, and RJMCMC. In every case, we can see those kind of optimisation as meta optimisation, where one level find optimal parameters values for a given number of parameters, and a second level find the optimal number of parameters (the two levels are intertwined). Therefore, without loss of generality, we focus on designing a method that find optimal parameters values first (first level). This method could be further used in more complex optimisation framework (second level).

Results and Forces visualisation

Visualising results directly within a GIS (QGIS in our examples) is extremely handy, as results can be compared to aerial view and other reference vector layers. Moreover, the GIS software is necessary anyway for the user to input manual kerb point and/or correct automatic observations. Currently the optimisation software is independent, but it could be integrated as a plug-in, or automatically triggered when user modifies the observations for instance (See Chapter4 for more in base-interactive behaviour).

Optimisation results

Result evaluation

"PARIS" EXPERIMENT AREA The result (Table 8) at Paris scale ("Paris") with ground truth eliminate observation bias. Without constraints, inputing ground-truth as observation, we get in the 0.1m range from ground truth. We explain this remaining distance by the limit of expressiveness of the road model (no asymetric width, no linear varying width), and more casually, by the ground truth own faults (large avenue ground truth contains side-walk but also other road features), and by overconstraints (road axis are not split enough). Figure 129 illustrates those common limitations.

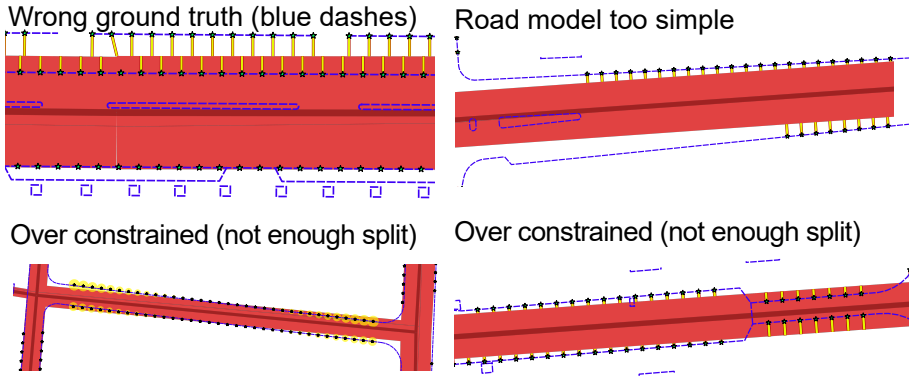


Figure 129: Various illustrations of optimisation process limitation. The data used as ground truth contains errors. Then road axis main not split enough. Last, the simple road model used can not deal with some features of Paris roadway.

All factors included, 0.1m is a very satisfactory result, as it would be about the same size as a precise aerial image pixel, or about Open Data Paris official precision.

Using forces to resist change and stay close to the initial road model is unnecessary when using reliable and dense observations. We could not find a place where regularisation forces were really useful. In a similar topic, optimisation appeared to be robust, even when the road model was obviously not suited to model the complex roads.

Of course we cannot expect to have access to observations with such a high quality and coverage for a real life use case.

"SENSING" EXPERIMENT AREA Results (Table 9) on both regular and alternate dataset confirm that a fully automatic process with kerb observation can reach 0.6m to ground truth distance, thus significantly improving the initial median distance (1.5m), even using quite noisy observations. The main reasons explaining the difference between this result and the result using observation from ground-truth is visually immediate: observation sparsity.

Given the object detection we dispose of, they are either too sparse (road signs, road markings) or too noisy (cars, other objects) to be really useful in a full automated process. They only marginally improve model fitting, at a high computational cost. However such object observations have a real potential, as proved by a simple experiment we performed. We manually created a complete marking dataset for a road. After optimisation, the model was a really good fit (parking place markings are especially useful, as well as correct pedestrian crossings). Similarly, markings from Hervieu, Soheilian, and Brédif, 2015 which are more complete delivered promising results.

"Sensing" area is fairly sized, making it representative. For the goal of evaluating the impact of road axis being over-constrained (not enough split), we automatically split the road axis so that no road segment is more than 10m long. This gives a fairly good certitude that road axis are no more over-constrained. The average distance to ground truth after optimising using ground truth observations is about 0.1m on *average*, with a median distance of 0.015m, without regularisation forces and excluding observation in intersection area (m) from our measures. Therefore, the only remaining factor explaining this residual is the fact that our road model is too simple to explain complex Paris

roads. All considering, this proves that even the very simple road model is sufficient to come very close to complex road surface of Paris.

"USER" EXPERIMENT AREA The reduced area is especially telling regarding road model limitation (Table 10), although this area is smaller so less representative. In this area, we can manually control that road axis are split into the correct number of segments, and that sidewalk ground truth only contains sidewalk. When using observation from ground truth, the only error factor remaining is then the road model limitations (See Fig. 118). Despite some very complex road features (asymmetric width, pedestrian island, non constant road), the model is very close to ground truth (0.1m).

Using sensing data on the area yields an interesting result, as it gives a realistic example of automatic fitting for a well split road axis network. Optimisation results are about 0.45m from ground truth side-walk. For this area we also tested a user-assisted approach, where user complement sensed kerb observation with manually added kerb points. The result is visually more acceptable, but quantitatively not very different. Indeed, results in this range hit the sensing vehicle geospatial precision, which is about 0.4m in this area where streets are narrow and buildings are high (masking GPS).

Experiments using object observations confirm that it would be hard to produce quality results using them. However kerb observations can be marginally complemented by object observations for a small diminution of average distance to ground truth.

Generating streets from optimised road model

Before generating streets with StreetGen, we regroup road segments with similar width and assign probable width to road segments that have no observations. In a sense this amount to a regularisation of road widths, and it should be performed directly during the optimisation process, using for instance the proposed width regularisation force.

StreetGen road model uses turning radius in intersection. These radius could also be determined by the optimisation process.

Concerning the topology errors, forces could be introduced within the optimisation process so a road segment can not intersect another one (except 1-neighbours). However such forces would have to be limited to N-neighbours, with a small N (for instance 3), so the number of forces is not too large.

CONCLUSION

Our goal was to find a way to model city roads so it better fits various observations (kerb, street objects, etc.) in a city surrounding, at various scale (whole city (e.g. Paris), few hundreds axis, few blocks), starting from the approximate road axis network with coarse road width. The road is modelled as a network of road segments with their own width. We formulate the problem as an optimisation problem aiming at finding the best road parameters (position of segment node, width values) that fits the observations.

Observations can be obtained automatically through sensing (street Lidar and images, aerial images), manually (user entered) and/or from legacy data (GIS vector layer describing side-walk). Observations are either consolidated kerb points detections, or street object detections, where each object type expected behaviour regarding the road is a setting (for instance pedestrian crossing markings are expected to be inside the

roadway surface). We consider the optimisation problem similarly to a mechanic optimisation problem, with observation exerting forces on road segments and width, and other opposing forces trying to keep close to initialisation values.

Finding the global optimal model to fit the observations is a complex task, as not only the position of road segments and their width have to be optimised, but also which observation affects what segment, and even the number of segments (creating/deleting/splitting/merging segments). In this work, we solve only a part of the optimisation problem within a non-linear least square framework, to find the road edge position and width optimal values. Initial model fitting is not very good, although reasonable (median distance between model and (not perfect) ground-truth side-walk is 1.8m).

The results with synthetic data (observation derived from ground truth side-walk) suggest that the optimisation process can produce a solution closely fitting the actual roads (median distance of 0.1m) at large scale quickly (e.g. whole Paris city in few minutes). This proves that even a simple road model is sufficient to model the vast majority of the complex Paris roadway.

Fully automatic results with real sensing data show that kerb detection can be used effectively to fit the model (median distance 0.6 m), especially when the road network segments are adequately split (median distance 0.45 m). The main factor explaining the remaining distance seems to be the lack of data, which can be complemented by user input (median distance of 0.35 m), and the inherent sensing data precision (about 0.3 m). Object observations have to be reasonably noisy or they are not as useful as kerb observation (median distance of 1 m), even if they can be used to slightly complement kerb observations.

Furthermore, our optimisation framework is fast enough (< 1 s)to be used in an interactive guided scenario when focusing on few roads (for editing or corrections for instance).

Promising and fast results make the proposed method suited to be integrated as a step into a more powerful optimisation framework, such as a Reversible Jump Markov Chain Monte Carlo method (RJMCMC, Green, 1995), which would be able to fully deal with observation outliers removal and adequate introduction/removal of new road segments, as well as different type of road segment (including one with linearly varying width).

Part III

CONCLUDING THE THESIS

6

A GENERAL CONCLUSION TO THIS THESIS

GENERAL CONCLUSION

Thesis work

In this thesis, we propose an original approach to reconstruct streets (street modelling). We argued that the Inverse Procedural Modelling (IPM) paradigm is well suited to street reconstruction.

Therefore we propose a simple and extensible procedural street generation tool (StreetGen, Chap. 3) generic enough to be used by many applications (visualisation, traffic simulation, urban planing, etc.). It requires very little information (a road axis network and a few attributes), that are conceptually obtainable (fully automatically or semi-automatically) from sensing data. We use a relational database management system (RDBMS) to both deal with inputs, perform the modelling (street generation) and store the results.

This database is further exploited to host in-base interactive edit (Chap. 4), a new paradigm of user edit where the handling of edit is not done in a custom software, but rather directly inside the database. This can be seen as a manual form of IPM. This approach allows to use any GIS software as a graphical interface provided it can read and write in a database, and brings an additional multi-user capacity.

We explore an automatic method to reduce the necessary user interaction by using optimisation together with sensing data.

First, large amount of Lidar point clouds are collected by a mobile mapping vehicle. Given the size and lack of organisation of those data, we introduce the Point Cloud Server (PCS, Chap. 2) to manage those. The PCS is an extension to a common database server which allows to store both points (compressed) and meta-data inside the database. Furthermore, those points are generalised and put in relation with other geospatial data (vector and raster database, sensing vehicle trajectory, etc.), enabling fast filtering and access to points. In-base point cloud Level of Detail (LOD) and classification (7) were also explored.

Those sensing data are leveraged by extracting urban feature detection, such as road markings and curbstones, which are consolidated and filtered into urban feature observations. We then fit StreetGen road model to the observations, interpreting this optimisation problem as a mechanical one (Chap. 5), and robustly solving it with non linear least square method. We further integrate optional user interaction to complement eventual gaps in sensing data, validate the results and possibly correct them.

Contribution

We stress that each chapter of this thesis can be read independently, detailed conclusions are available at the end of each chapter. Our first contribution is to use relational database to generate and store a street model (StretGen), which, along with efficient

geometrical computing, allowed us to model the whole city of Paris. Embedding data, generation algorithms and resulting modelling in a database is a new and powerful approach, with concurrency, robustness and scalability capabilities. This street model has successfully demonstrated its usefulness for several applications, such as 3D visualisation and traffic simulation.

Our second contribution is the in-base interaction concept. This new concept using long established tools shows great potential to easily create custom interaction behaviour that greatly accelerate user task. A decisive advantage is universality, as an interaction in base can then be transparently used (concurrently) by many GIS software, custom applications and web clients.

This strategy was successfully demonstrated on many generic interaction example, and can be easily adapted, even in complex case. It proved to be powerful enough to handle and speed up all interactions for StreetGen.

Our third contribution is a complete solution to manage point clouds in a database server (Point Cloud Server). More than storing compressed groups of points, we demonstrated how all aspect of point cloud usage can be covered, with fast input and output, point generalisation and level of details, use of metadata and other geospatial data for filtering, as well as in-base and out of base processing.

Our last contribution is a complete workflow from observations to a city model and the demonstration that sensing data can be effectively be leveraged to fit a road model at city scale.

Thesis limitations and perspectives

The street modelling solution we propose has several limitations and can be improved in several interesting ways.

PROCEDURAL MODEL. The street model we use is simple, it cannot, for instance, model roads with asymmetric width (for instance for parking places). There is still no relation between objects, and then no composition of objects which are key for real life complex street objects. Moreover, the street model is based on a road axis network that cannot deal with bridges or tunnels. An interesting perspective to address these issues is to use a grammar (for instance shape grammar) in the street model. This would bring much more complexity and *expressiveness to the procedural model*, especially to manage advanced patterns of objects.

INVERSE PROCEDURAL MODELLING. The road model used in the optimisation scheme presented in this thesis is very limited, which probably slows optimisation and lowers result quality. The road model could be easily extended to better represents Paris street (varying width, asymmetric width). We could also use more diverse observations from more sensing data. Most importantly, the proposed optimisation cannot change the road network topology or remove (or ignore) erroneous observations. We could integrate our optimisation method into a more complex optimisation framework to solve this problem.

INVERSE PROCEDURAL MODELLING FOR GRAMMAR If StreetGen street model was extended using grammars, a different optimisation strategy could even be possible,

where a street model based on a street grammar is directly fitted to observations using a powerful optimisation framework such as RJMCMC (Green, 1995). In this case, the optimisation problem would become to find the optimal instantiation of a street grammar that best represents the observations of urban features.

USER INTERACTION. Interactions become hard to control when a large number of potential interactions are used together. Moreover, in our proposal, the possibilities to change the road network are limited. Nevertheless, in-base interactions are very generic, and could be complemented by more adhoc interfaces, such as plugins coupling classical GIS view (2D) and modern 3D streetviews (as prototyped in Sec. 4.5.7). This opens new perspectives for the usability of such tools for a wider community of users as well as new interaction approaches for the modelling of complex objects such as cities.

POINT CLOUD SERVER. In this thesis, we only scratch the surface of in-base processing with the Point Cloud Server, and some of those processings have not been demonstrated on a full dataset. In the future, the Point Cloud Server could be abstracted into a Point Cloud Service, a standard way to provide points, and possibly process them with standardized processing, in a similar spirit to WFS and WPS. In particular, this direction of research would promote the interoperability of such data sources.

SCALING We demonstrated the generation of a complex city (Paris), yet, it could be interesting to use StretGen on a much larger scale, such as on a whole country.

CITY MODEL Our methods could be combined with other reconstruction methods dedicated to buildings, specific urban furnitures, vegetations, etc., to create a full city model that could be used for many applications related to city. We suppose that in the long run, given that most of Mankind is going to live in cities, Artificial Intelligence system would also need this city model as their task would require them to interact with citizens.

Part IV

APPENDIX

7

APPENDIX : IMPLICIT LOD, DIMENSIONALITY DESCRIPTOR AND PATCH CLASSIFICATION FOR POINT CLOUD

The end goal fo this thesis is to (semi) automatic street reconstruction, which requires data. To this end, we use street Lidars point clouds, which form rich but very huge datasets (Billions of points). We manage this Lidar dataset by using a Point Cloud Server (Chap. 2). One of the key to use those Billion of points is to be able to work on only a small part of it. To this end, we used two strategies. The first is explored in Chapter 2, and amount to filtering the point clouds (potentially using complex criterias using other data such as vector or raster). However even after filtering, the resulting number of points may be too large because Lidar Point Cloud are extremely dense (about 1 kpts /m²).

Reducing point cloud density can be easily done by random subsampling, yet this type of subsampling can greatly reduce point cloud quality and usefulness. Therefore we explored another way to reduce number of points by using a Level Of Details (LOD) strategy. This LOD has been used in various point cloud processing methods of Chapter 2.

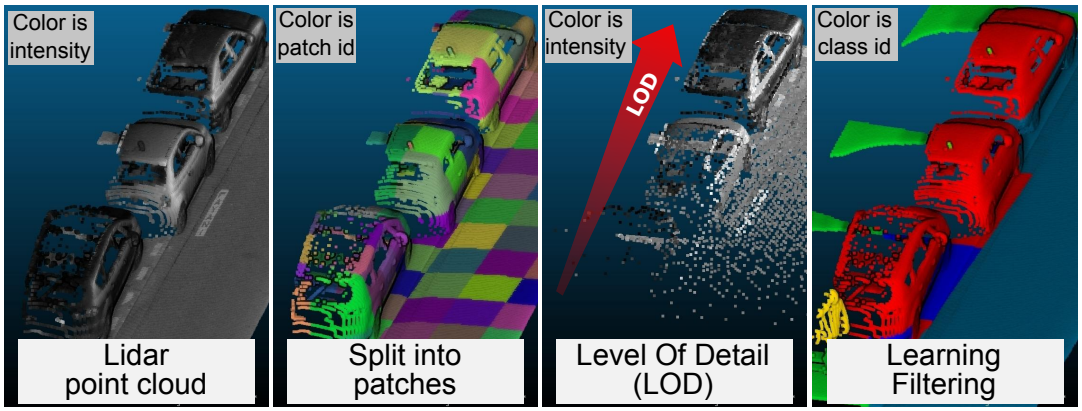


Figure 130: Graphical Abstract : a Lidar point cloud (1), is split it into patches (2) and stored in the PCS (Cura, Perret, and Paparoditis, 2015b), patches are re-ordered to obtain free LOD (3) (a gradient of LOD here), lastly the ordering by-product is a multiscale dimensionality descriptor used as a feature for learning and efficient filtering (4).

Lidar datasets now commonly reach Billions of points, as their density is often very large. Using these point cloud becomes challenging, as the high number of points is untractabel for most applications and for visualisation. In this work we propose a new paradigm to easily get a portable geometric Level Of Details (LOD) inside a Point Cloud Server. The main idea is to not store the LOD information in an external additional file, but instead to store it implicitly by exploiting the order of the points. The point cloud is divided into groups (patches). These patches are ordered so that theim in order gradually provides more and more details on the patch. We demonstrate the interest of our method with several classical uses of LOD, such as visualisation of massive point cloud, algorithm acceleration, fast density peak detection and correction. Furthermore, our implicit LOD method also embeds information about the sensed object geometric nature, forming a crude multi-scale dimensionality descriptor. We demonstrate the interest of this descriptor by successfully using it for fast classification(basic classes) and on-the-fly filtering.

Problem

Democratisation of sensing device have resulted into an expansion of acquired point clouds. In the same time, acquisition frequency and precision of the Lidar device are also increasing, resulting in an explosion of number of points.

Datasets are now commonly in the multi billion point range, leading to practical issue to store and use them. Moreover, point cloud data usage is more common and no more limited to a specialized community. Non specialised users require easy access to data. By necessitating easy access and storage and processing for a large amount of data, point clouds are entering the Big Data realm.

Yet all those data are not always needed; having the complete and fully detailed point cloud is impracticable, unnecessary, or even damageable for most applications. Therefore, the ability to reduce the number of points is a key point for practical point cloud management and usages.

The number of points must not only be reduced, but often the density corrected. Indeed, point clouds from Lidar do not have a constant density. The sensing may be structured for the sensing device (for instance a Lidar may sense point using a constant angle), but not necessary for the sensed object (see Fig. 131). Furthermore,fusing multiple point clouds also produce non regular density.

Figure 131: Regular sensing does not imply regular sampling.

There are basically two approaches to reduce the amount of data considered (See Figure 132). The first is to use a **filtering** strategy based on data characteristics (position, time, semantic, etc.) which keeps only a portion the original data. The second is a **generalisation** strategy, where we replace many points with fewer objects that represent appropriately those points. For instance, in order to visualize massive point cloud, it's important to fetch only the appropriate points by selecting the ones which are visible (filtering) and which are the most representative of the scene (generalisation) at the same time.

Many methods perform filtering, usually by using simple spatial criteria (for instance, points in polygon). Generalisation is also popular in its most basic form (generalise points by points). Cura, Perret, and Paparoditis, 2015b covers extensively filtering with many possibilities (spatial, semantic, attributes, using vector and raster data, using metadata), and also proposes generalisation. Nevertheless it uses a generalisation approach only based on more abstract types (bounding box, planes, stats, etc.), which limits its use to methods that are adapted to those types. It does not generalise points by points.

In this work we propose to extend the PCS to explore the generalisation of groups of points by choosing a representative subset of points (See Fig. 132).

We propose to use Level Of Details that reduce successively the number of points while preserving the geometric characteristics of the underlying sensed object. Our method is designed to be efficient, robust to point density variation and can be used for many large point clouds processing, including visualisation.

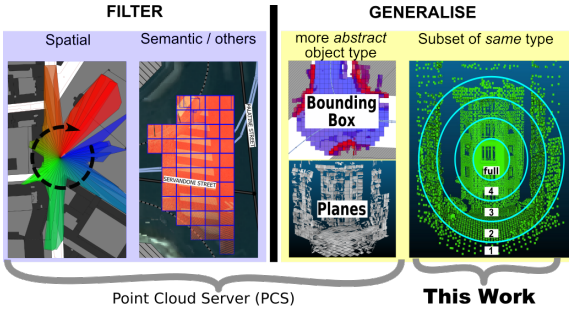


Figure 132: Two strategies to limit the amount of points to work on.

Related Work

Finding a subset of point that represents well all the points is a common problem. It has been extensively studied in Geographical Information System (GIS)and other research field. It could be seen as compression, clustering, dimensionality reduction, or Level Of Detail (LOD) for visualisation.

Sophisticated methods have been proposed to generalise 2D points for cartographic applications (Sester, 2001, Schwartges et al., 2013). Yet those methods are limited to 2D points, and could not be easily modified to work in 3D. Indeeed, those methods are cartographic by nature, which means that they rely on having all the points on a simple surface : the 3D plan formed by the map. Applying directly such methods to point clouds would thus require to have access to surfaces of sensed objects. Yet, getting this surface (reconstruction) is a very hard challenge, sometime without solution, and thus we can not rely on it. For those limitation and large computing cost, those advanced methods can not be used for large 3D point clouds.

Other much simpler methods have been designed to work on 3D points. Because the goal is to produce hierarchical levels of points, it seems natural to use a hierarchical structure to compute those levels. The main idea is to build a hierarchy of volumes, then each level of the hierarchy corresponds to a LOD. For each volume, a point is created/chosen to generalise all the points within the volume. Rusinkiewicz and Levoy, 2000 use a Bounding Sphere Hierarchy for a visualisation application. Yet spheres are not well adapted to represent planes, which form a large part of man-made objects and structures. On the other hand, Octree (Meagher, 1982) have become the de-facto choice. It seems that the most popular use of Octree is as spatial acceleration structure (spatial index). Octree have several advantages. The first is that their basic nature is closely related to Morton (or GeoHash) order, making them efficient to build (Sabo et al., 2014, Feng and Watanabe, 2014). They can also be created out of memory for extremely large point clouds (Baert, Lagae, and Dutré, 2014). Moreover, their regularity allows efficient representation and compression (Huang et al., 2006; Schnabel and Klein, 2006), as well as fast geospatial access to data (Elseberg, Borrmann, and Nüchter, 2013).

Octree are also natural condidates to nesting (i.e. create a hierarchy of octrees with various resolution and occupancy, as in Hornung et al., 2013). Octree construction into file system hierarchy approach is still popular today (Oscar Martinez-Rubi et al., 2015), with point cloud in the 600 Billions points range. It has also been adapted to distributed

file system (cloud-computing) [1], with processing of 100 Billions points at 2 Billions pts /hour using a 32 cores 64 GB computer.

However, the method using Octrees present several disadvantages. Each method uses a custom octree format that is most often stored in an external file. This raises problems of concurrency and portability.

There a several ways to use an Octree to generalise points. We could not find a study of those ways for 3D points. However, Bereuter, 2015 recently gave an overview of how quad tree can be used for point generalisation. Quad trees are 2D Octrees, yet Bereuter, 2015 analyse can be directly translated in 3D.

The steps are first to compute a tree for the point cloud. Then, the point generalisation at a given level is obtained for each cell of the same tree level, by having one point represent all the points of this cell.

There are two methods to choose a point representing the others. The first one is to select on points among all ('select'). The second method is to create a new point that will represent well the others ('aggregate'). Both these methods can use geometry of points, but also other attributes.

In theory, choosing an optimal point would also depend on application. For instance lets consider a point cloud containing a classification, and suppose the application is to visually identify the presence of a very rarely present class C. In this case a purely geometrical LOD would probably hide C until the very detailed levels. On the opposite, prefering a point classified in C whenever possible would be optimal for this application.

However, a LOD method has to be agnostic regarding point clouds, and point clouds may have many attributes of various type and meaning, as long as many applications. Therefore, most methods use only the minimal common factor of possible attributes, that is spatial coordinates. For visualisation applications, aggregating points seems to be the most popular choice Elseberg, Borrmann, and Nüchter, 2013; Hornung et al., 2013; Schütz and Wimmer, 2015. with aggregating functions like centroids of the points or centroid of the cell.

All of this methods also use an aggregative function (barycentre of the points, centroid of the cell) to represent the points of a cell. Using the barycentre seems intuitive, as it is also the point that minimize the squared distance to other points in the cell, and thus a measure of geometric error.

However, using the 'aggregate' rather than 'select' strategy necessary introduces aggregating errors (as opposed to potential aliasing error), and is less agnostic. Indeed, aggregating means fabricating new points, and also necessitate a way to aggregate for each attributes, which might be complex (for instance semantic aggregating; a point of trash can and a point of bollard could be aggregated into a point of street furniture). This might not be a problem for visualization application. Yet our goal is to provide LOD for other processing methods, which might be influenced by aggregating errors. Furthermore, the barycentre is very sensible to density variations.

Therefore, we prefer to use a 'select' strategy. The point to be selected is the closest to the centroid of the octree cell. If the point cloud density is sufficient this strategy produces a nearly regularly sampled point cloud, which might be a statistical advantage for processing methods. To establish a parallel with statistics, picking one point per cell is akin to a Latin Hypercube (see McKay, Beckman, and Conover, 1979). Avoiding the

1 https://github.com/connormanning/entwine

averaging strategy might also increase the quantity of information than can be retrieved (similar to compressed sensing, see Fornasier and Rauhut, 2010).

We note that most of the LOD systems seems to have been created to first provide a fast access to point (spatial indexing), and then adapted to provide LOD. Using the PCS, we can separate the indexing part, and the LOD scheme. From this stems less design constraints, more possibilities, and a method than is not dedicated to only one application (like visualisation).

Contribution

This work re-uses and combines existing and well established methods with a focus on simplicity and efficiency. As such, all the methods are tested on billions scale point cloud, and are Open Source for sake of reproducibility test and improvements

- In (Section 7.3.2) is to store the LOD implicitly in the ordering of the points rather than externally, avoiding any data duplication. Thus, we don't duplicate information, and the more we read points, the more precise of an approximation of the point cloud we get. If we read all the points, we have the original point cloud.
- We introduce (MidOc, Section 7.3.3), a simple way to order points in order to have an increasingly better geometric approximation of the point cloud when following this order.
- In (Section 7.3.4) we show that this ordering embed information about the dimensionality of the sensed object, to the point of being a simple multi-scale dimensionality descriptor. We demonstrate the interest of this descriptor by comparing it to a state of the art dimensionality descriptor, then by assessing it potential by performing a Random Forest classification that can then be used for very fast pre-filtering of points, and other applications.

Plan

This work follows a classical plan of Introduction Method Result Discussion Conclusion (IMRAD). Section 7.3 presents the LOD solution, how it produces a dimensionality descriptor, and how this can leveraged for classification. Section 7.4 reports on the experiments validating the methods. Finally, the details, the limitations, and potential applications are discussed in Section 7.5.

METHOD

In this section, we first present the Point Cloud Server (section 7.3.1)(PCS Cura, Perret, and Paparoditis, 2015b) that this article extends. Then we introduce the LOD solution that we propose , which consists of reordering groups of points from less to more details (7.3.2), and then choose which LOD is needed. Although any ordering can be used, we propose a simple geometric one (7.3.3) which is fast and robust to density variation. Furthermore, constructing this ordering produces a rough dimensionality descriptor (7.3.4). This descriptor can be used in the PCS to perform density correction (7.3.5) and

classification at the patch level (7.3.6). This classification can be directly transferred to points, or indirectly exploited in a pre-filtering step.

The Point Cloud Server

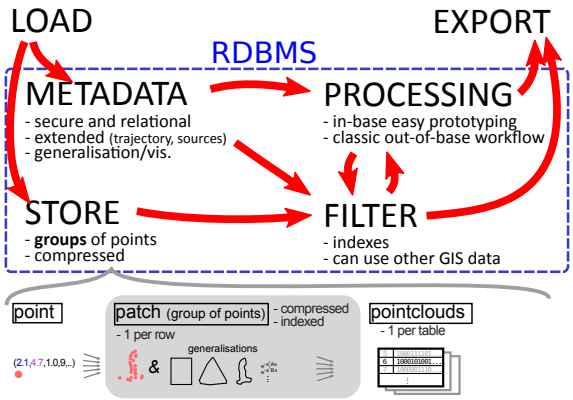


Figure 133: Overall and storage organisations of the Point Cloud Server.

Our method strongly depends on using the Point Cloud Server described in Cura, Perret, and Paparoditis, 2015b, therefore we introduce its principle and key relevant features (see figure 133).

The PCS is a complete and efficient point cloud management system based on a database server that works on groups of points rather than individual points. This system is specifically designed to solve all the needs of point cloud users: fast loading, compressed storage, powerful filtering, easy data access and exporting, and integrated processing.

The core of the PCS is to store groups of points (called patches) that are multi-indexed (spatially, on attributes, etc.), and represented with different generalisation depending on the applications. Points can be grouped with any rules. In this work, the points are regrouped spatially by cubes 1m (Paris) or 50m (Vosges) wide.

All the methods described in this work are applied on patches. We propose is to reorder each patch following the MidOc ordering, allowing LOD and producing a dimensionality descriptor per patch. It can then be used to classify patches.

We stress that our method used on any point cloud will provide LOD, but that using it with the PCS is much more interesting, and adds key feature such as spatial indexing, fast filtering, etc.

Exploiting the order of points

We propose to exploit the ordering of points to indirectly store LOD information. Indeed, whatever the format, be it file or database based, points ends up as a list, which is ordered.

The idea is then to exploit the order of this list, so that when reading the points from beginning to end, we get gradually a more accurate geometrical approximation of the point cloud (see figure 134).

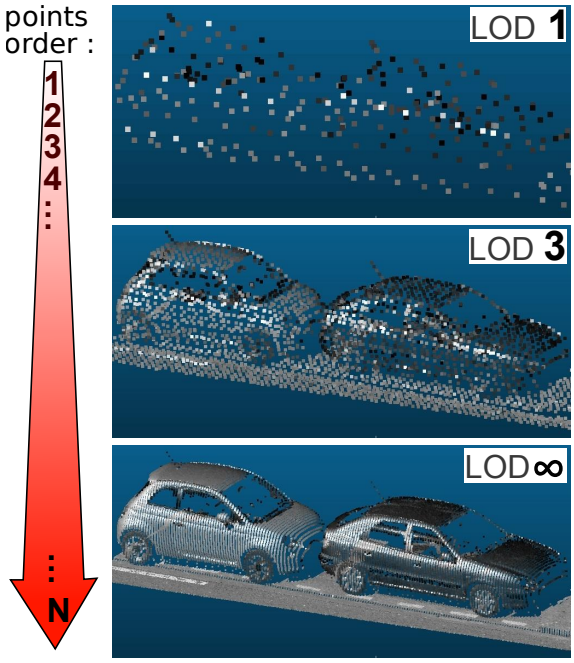


Figure 134: 3 Geometrical Level Of Detail (LOD) for the same point cloud. Reading points from 1 to N gradually increases the details, because of the specific order of points (MidOc).

For instance, given list $L[P_1, .., P_N]$ of ordered points. Reading P_1 to P_5 gives a rough approximation of the point cloud, and reading another 16 points (P_1 to P_{21}) is going to give a slightly better approximation. Reading points 1 to N is going to get the exact point cloud, so there is no data loss, nor data duplication.

Using the point ordering as LOD results in three main advantages.

IMPLICIT Except a pre-processing step to write the point cloud following a given ordering, each time the user wants to get a Level Of Detail version of the point cloud, there is no computing at all (only data reading). This may not make a big difference for non-frequent reading, but in a context where the same point cloud is going to get used several times at several levels and by several users simultaneously (for instance Point Cloud as a Service), no processing time makes a big difference.

NO DUPLICATION Another big advantage is that exploiting point ordering does not necessitate additional storage. This is an advantage on low level. It saves disk space (no data duplication, no index file). Because the LOD information is embedded right within the point cloud, it is perfectly concurrent-proof, i.e. the point cloud and the LOD can not become out of sync. (Even in heavy concurrent Read/Write, a user would always get a coherent LOD). Lastly because the LOD only relies on ordering the original points, and does not introduces any other points or data, it avoids all precision-related issues that may come from aggregating.

PORTABLE The last advantage comes from the simplicity of using the ordering. Because it is already something that all point cloud tools can deal with (a list of points!), it is portable. Most softwares do not change the points order inside a cloud (See Section

7.4.3). Even if a tool were to change the order, it would be easy to add the ordering number as an attribut (though slightly increasing the storage requirement). This simplicity also implies that adapting tools to exploit this ordering is very easy.

MidOc : an ordering for gradual geometrical approximation

Requirements and hypothesis

The method exploits the order of points to store LOD information, so that the more points are read, the more detailed the result becomes. Obviously an ordering method that class the points from less details (LOD_0) to full details(LOD_∞) is needed. This ordering is in fact a geometric measure of point relevance, that is how well a point represents the point cloud (in a neighbourhood depending of the LOD).

This ordering will be used by on different point clouds and for many applications, and so can not be tailored to one. As such, we can only consider the geometry (the minimal constituent of a point). Because of the possible varying-density point clouds, the ordering method also have to recreate a regular-ish sampling.

Although many ordering could be used (for example, a simple uniform-random ordering), a suitable one would have low-discrepancy (that is be well homogeneous in space, see Rainville et al., 2012), not be sensitive to density variations, be regular, be fast to compute and be deterministic (which simplify the multiuser use of the point cloud).

We make two hypothesis that are mostly verified on Lidar point cloud. The first hypothesis ('disposable density') is that the density does not gives information about the nature of the object being sensed. That is, depending on the sensing situation, some parts of the cloud are more or less dense, but this has nothing to do with the nature of the object sensed, thus can be discarded. The second hypothesis (low noise) is that the geometrical noise is low. We need this hypothesis because 'disposable density' forbids to use density to lessen the influence of outliers.

A common method in LOD is to recursively divide a point cloud into groups and use the barycentre of the group as the point representing this group. The ground of this method is that the barycentre minimise the sum of squarred distance to the points.

However such method is extremely sensible to density variation, and artificially creates new points.

Introducing the MidOc ordering

We propose the re-use of well known and well proven existing methods that is the octree subsampling (for instance, the octree subsampling is used in Girardeau-Montaut, 2014). An octree is built over a point cloud, then for each cell of the octree the LOD point is the barycentre of the points in the cell. With this, browsing the octree breadth-first provides the points of the different levels.

We adapt this to cope with density variation, and to avoid creating new point because of aggregation. We name this ordering MidOc (Middle of Octree subsampling) for clarity, nonetheless we are probably not the first to use it.

The principle is very simple, and necessitate an octree over the point cloud (octree can be implicit though). We illustrate it on Figure 135 (in 2D for graphical comfort). We walk the octree breadth-first. For each non-empty cell, the point closest to the cell centre is chosen and assigned the cell level, and removed from the available point to

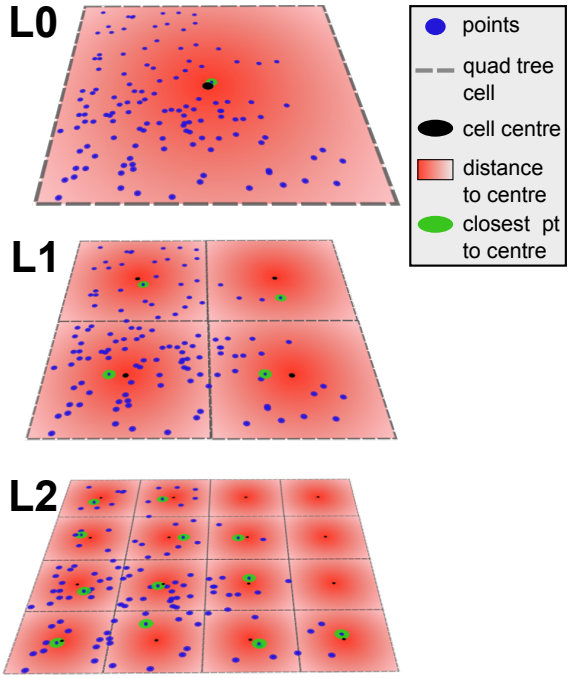


Figure 135: MidOc explained in 2D. Given a point cloud (Blue) and quad tree cells (dashed grey), the chosen point (green ellipse) is the one closest to the centre (black point) of the cell.

pick. The process can be stopped before having chosen all possible points, in which case the remaining points are added to the list, with the level L_∞.

The result is a set of points with level (P, L_i). Inside one level L_i, points can be ordered following various strategies (see Section 7.3.3.4).

Because each point is assigned a level, we can store the total number of points per level, which is a multi-scale dimensionality descriptor, see Section 7.3.4).

Implementation

MidOc ordering is similar to octree building. Because Octree building has been widely researched, we test only two basic solutions among many possibilities.

The first kind of implementation uses SQL queries. For each level, we compute the centres of the occupied cells using bit shifts and the closest point to these. Picked points are removed, and the process is repeated on the next level. It relies on the fact that knowing each point octree cell occupancy does not require to compute the octree (see Figure 150).

The second implementation uses python with a recursive strategy. it only necessitates a function that given a cell and a list of points chose the point closest to the centre of the cell, then split the cell and the list of points for the next level, and recursively calls itself on this subcells with the sublists.

A more efficient and simpler implementation is possible by first ordering the points following the Morton (Hypothesis : or Hilbert) curve, as in Feng and Watanabe, 2014 (Section 2.5.1, page 37), in the spirit of linear octree.

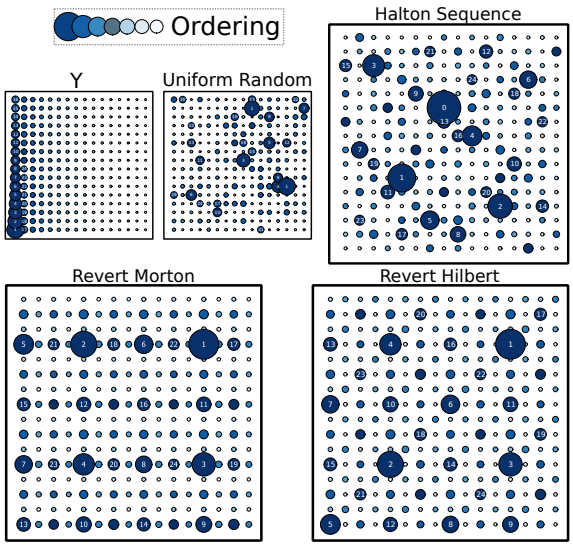


Figure 136: Several possible intra-level orders with various coverage from bad to good. Revert Morton and Revert Hilbert have offset for illustration.

Inside one LOD points can be ordered with various methods. The intra-level ordering will have an impact if the LOD is used in a continuous way, and moreover may influence methods that relies on low-discrepancy. More precisely, if only a part of the points in a level are going to be used, it may be essential that they cover well the spatial layout of the totality of points. Several methods give this kind of coverage (see Rainville et al., 2012)

Lets take the example where the goal is to find the plan that best fits a set of points and the method to do so in online (for instance it could be based on online robust PCA like in (Feng, Xu, and Yan, 2013)). The plan fitting method reads one by one the points of a designated level L_i, and successively computes a better plan estimation.

The Figure 136 presents some possible ordering. If the plan detection method was applied on the Y ordering, it would necessitate a great number of points to compute a stable plan. For instance the first 16 points (1 column) would not permit to compute a plan. Similarly, if the point were ordered randomly, estimating a plan would still require lots of points, because uniform randomly chosen points are not well spread out (on the figure, the first 25 points are over represented in the upper left part).

On the opposite, using a low discrepancy ordering like the Halton sequence makes the points well spread, while being quasi-random. Inverted space filling curves like the Morton or Hilbert curves also cover well space, at the price of being much more regulars.

The Halton sequence ordering is obtained by generating a Halton sequence (nD points) and successively pick points closest to the Halton generated points. The revert Morton ordering and revert Hilbert ordering are the distance along Morton or Hilbert curve expressed in bit and read backward (with a possible offset).

Crude multi-scale dimensionality descriptor (MidOc by-product)

Principle

During MidOc building process, the number of chosen points per level can be stored. Each ordered patch is then associated with a vector of number of points per level $\mathtt{ppl} = (N_{L_1}, .., N_{L_{max}})$. The number of picked point for L_i is almost the voxel occupancy for the level i of the corresponding octree. Almost, because in MidOc points picked at a level do not count for the next Levels. Occupancy over a voxel grid has already been used as a descriptor (See Bustos et al., 2005). However we can go a step further. For the following we consider that patches contain enough points and levels are low enough so that the removing of picked points has no influence.

In theory for a level L_i, a centred line would occupy 2^{L_i} cells, a centred plan 4^{L_i} cells, and a volume all of the cells (8^{L_i} cells). Thus, by comparing $\mathtt{ppl}[L_i]$ to theoretical $2^{L_i}, 4^{L_i}, 8^{L_i}$ we retrieve a dimensionality indice $\mathtt{Dim}_{\mathtt{LOD}}[i]$ about the dimensionality of the patch at the level L (See Figure 137). This occupancy is only correctly estimated

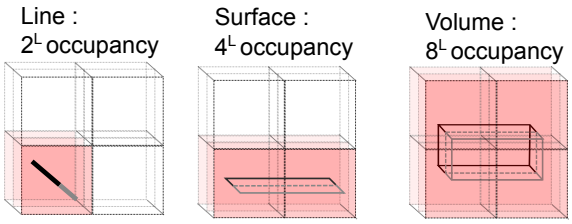


Figure 137: Voxel occupancy is a crude dimensionality descriptor: 3D line, surface or volume occupy a different amount of voxels.

when the patch is fully filled and homogeneous. However, we can also characterize the dimensionality $\mathtt{Dim}_{\mathtt{LODDiff}}$ by the way the occupancy evolves (difference mode). Indeed, a line occupying k cells of an octree at level L_i will occupy $2 * k$ cells at the level L_{i+1}, if enough points.

We stress that the information contained in $\mathtt{ppl}$ is akin to a multi-scale dimensionality indice, with the scale being the level of the octree. For the rest of this work we consider that the dimensionality is roughly the same across level (which is entirely false in some case, see 7.4.6.1).

The Figure 138 illustrate this. Typical parts of a street in the Paris dataset were segmented: a car, a wall with window, a 2 wheelers, a public light, a tree, a person, poles and piece of ground including curbs.
Due to the geometry of acquisition and sampling, the public light is almost a 3D line, resulting in the occupation of very few octree cells. A typical number of points chosen per level for a public light patch would then be $(1, 2, 4, 8, ...)$, which looks like a 2^L function. A piece of ground is often extremely flat and very similar to a planar surface, which means that the points chosen per level could be $(1, 4, 16, 64...)$, a 4^L function. Lastly a piece of tree foliage is going to be very volumetric in nature, due to the fact that a leaf is about the same size as point spacing and is partly transparent to laser (leading to several echo). Then a foliage patch would typically be $(1, 8, 64...)$ (if enough points), so a 8^L function. (Tree patches are in fact a special case, see 7.4.6.1).

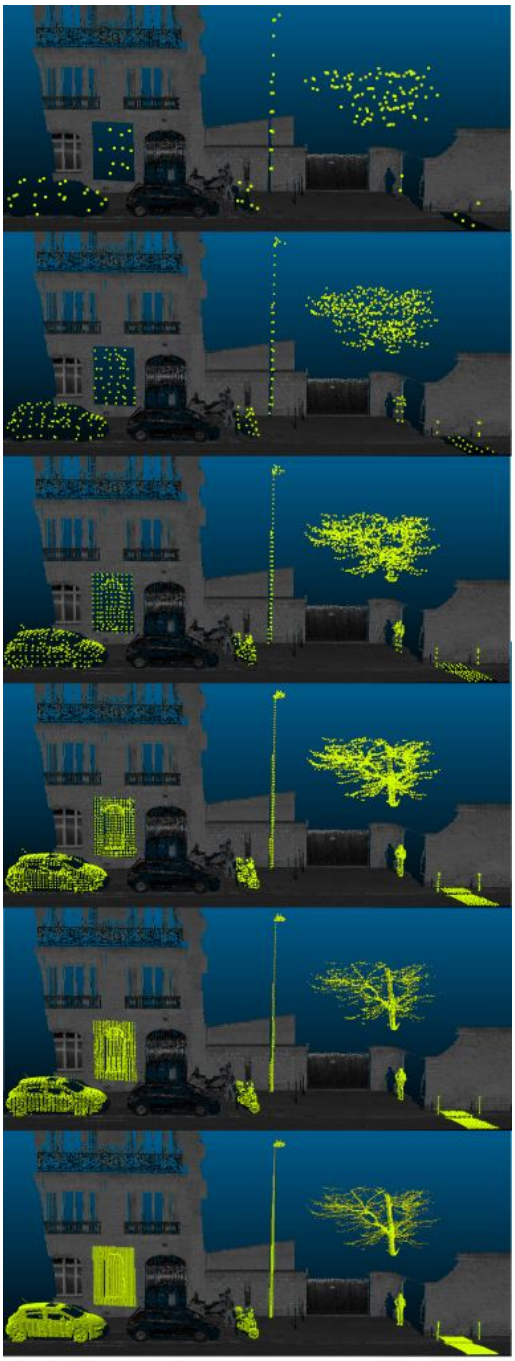


Figure 138: All successive levels for common objects (car, window, 2 wheelers, light, tree, people, pole, ground), color is intensity for other points.

Comparing crude dimensionality descriptor with covariance - based descriptors

A sophisticated per-point dimensionality descriptor is introduced in Demantké, 2014; Weinmann et al., 2015, then used to find optimal neighbourhood size. A main difference is that this feature is computed for each point (thus is extremely costly to compute), and that dimensionality is influenced by density variation.

At the patch level, we do not need to find the scale at which compute dimensionality, the descriptor is computed on the whole patch.

This dimensionality descriptors ($\mathtt{Dim}_{cov}$) relies on computing covariance of points centred to the barycentre (3D structure tensor), then a normalisation of covariance eigen values. As such, the method is similar, and has the same advantages and limitation, as the Principal Component Analysis (See Shlens, 2014 for a reader friendly introduction). It can be seen as fitting an ellipsoid to the points.

First this method is sensible to density variations because all the points are considered for the fitting. As opposite to our hypothesis (See Section 7.3.3.1), this method considers implicitly that density holds information about the nature of sensed objects. Second, this methods only fits one ellipse, which is insufficient to capture complex geometric forms. Last, this method is very local and does not allow to explore different scale for a point cloud as a whole. Indeed this method is classically used on points within a growing sphere to extend the scale.

However scale should be defined as the size of the features being analysed in sensed objects, and ot the scale of the neighbourhood of the centroid.

We compute both dimensionality descriptor and then compare them for the Paris dataset.

crude dimensionality descriptor as a feature

Using the result of the MidOc ordering has the advantage of not necessitate extra computing, the patch being ordered with MidOc for LOD anyway.

Moreover, because $x_1 \rightarrow (2^1)^x$, $x_2 \rightarrow (2^2)^x$, $x_3 \rightarrow (2^3)^x$ diverge very fast, we only need to use few levels to have a quite good descriptor. For instance, using $L = 2$, we have $x_i = [4, 16, 64]$, which are very distinguishable values, and don't require a total density above 70 points per patch. As long as the patch contains a minimal number of points, the descriptors is density and scale invariant. Lastly a mixed result (following neither of the $x_i \rightarrow (2^i)^x$ function) can be used as an indicator that the patch contains mixed geometry, either due to nature of the objects in the patch, or due to the way the patch is defined (sampling).

Although it might be possible to go a step further and decompose a patch $\mathtt{ppl}$ vector on the base of $x_i \rightarrow (2^i)^x, i \in [1..3]$, the direct and exact decomposition can't be used because the decomposition might depends on L_i. For instance a plane with a small line could appear as a plan for L_1 and L_2, and starts to appear differently over L_3 and higher level. In this case, an Expectation-Maximization scheme might be able to decompose robustly.

Excessive Density detection and correction

Lidar point cloud do not have a constant density, even if the acquisition is performed at a constant sensing rate, because the sensed object geometry (See Fig. 131).

Important variation of density can be a serious issue for some processing methods. For instance if millions of points are concentrated in a small volume, a processing method operating on fixed size volume may exceed the maximum memory of the system. Large density variation are also bad for performances in parallel environment. Indeed, efficient parallel computing may require that all the workers have about the

same amount of work. One worker stumbling upon a very dense part of the point cloud would have much more points to process than the other workers. The figure 141 shows a place in the Paris dataset where the density is 5 times over the average value of this data set. In this context of terrestrial Lidar, this density peak is simply due to the fact that the acquisition vehicle stopped at this place , while continuing to sense data.

The PCS coupled with LOD patches allows to quickly find abnormally high density. The PCS filters in few milliseconds the patch containing lots of points. This suffice for most applications. For a finer density estimation, we compute the approximate volume of the patch. For a level L, the ppl[L] number of points multiplied by the theoretical cell size for this level gives an approximate volume (or surface) of the patch. The total number of points divided by this volume (surface) gives a finer volumetric (surface) density estimation.

Then, correcting density consists of taking into account only the first K points, where K is computed to attain the approximate patch volume (surface).

Classification with the Point Cloud Server

Principle

We propose to perform patch classification using the Point Cloud Server and the previously introduced crude multi-scale dimensionality descriptor, along with other basic descriptors, using a Random Forest classifier. Following the position of the PCS towards abstraction, the classification is performed at the patch level and not at the point level. This induces a massive scaling and speeding effect, at the cost of introducing quantization error. Indeed, compared to a point classification, a patch may contain points belonging to several classes (due to generalisation), yet it will only be classified in one class, thus the "quantization" error.

Because patch classification is so fast and scales so well, the end goal can be however slightly different than for usual point classification.

Patch classification can be used as a fast preprocess to another slower point classification, be it to speed it (an artificial recall increase for patch classification may be needed, see Figure 149), or to better a point classification. The patch classification can provide a rough classification. Based on that the most adapted point classifier is chosen (similarly to Cascaded classifiers), thus improving the result of the final point classification. For instance a patch classified as urban object would lead to chose a classifier specialized in urban object, and not the general classifier. This is especially precious for classes that are statistically under-represented.

Patch classification may also be used as a filtering preprocess for applications that only require one class. Many applications only need one class, and do not require all the points in it, but only a subset with good confidence. For this it is possible to artificially boost the precision (by accepting only high confidence prediction). For instance computing a Digital Terrain Model (DTM) only requires ground points. Moreover, the ground will have many parts missing due to objects, so using only a part of all the points will suffice anyway. The patch classifier allow to find most of the ground patch extremely fast. Another example is registration. A registration process typically require reliable points to perform mapping and registration. In this case there is no need to use all points, and the patch classification can provide patches from ground and façade

with high accuracy (for point cloud to point cloud or point cloud to 3D model registration), or patches of objects and trees (for points cloud to landmark registration). In other applications, finding only a part of the points may be sufficient, for instance when computing a building map from façade patches.

Random Forest method started with Amit and Geman, 1997, theorized by Breiman, 2001 and has been very popular since then. They are for instance used by Golovinskiy, Kim, and Funkhouser, 2009 who perform object detection, segmentation and classification. They analyse separately each task on an urban data set, thus providing valuable comparison. Their method is uniquely dedicated to this task, like Serna and Marcotegui, 2014 (in print) who provide another method and a state of the art of the segmentation/-classification subject. Both of this methods are in fact 2D methods, working on an elevation image obtained by projecting the point cloud. However we observe that street point clouds are dominated by vertical surfaces, like building (about 70% in Paris data set). Our method is fully 3D and can then easily be used to detect vertical object details, like windows or doors on buildings.

Classification details

FEATURES The first descriptor is ppl, the crude multi-scale dimensionality descriptor produced by the MidOc ordering (see Section 7.3.4). We use the number of points for the level $[1..4]$. For each level L, the number of points is normalized by the maximum number of points possible (8^L), so that every feature is in $[0, 1]$.

We also use other simple features that require very limited computing (avoiding complex features like contextual features). Due to the PCS patch compression mechanism, min, max, and average of any attributes of the points are directly available. Using the LOD allows to quickly compute other simple feature, like the 2D area of points of a patch (points being considered with a given diameter).

DEALING WITH DATA SET PARTICULARITIES The Paris data set classes are organized in a hierarchy (100 classes in theory, 22 populated). The rough patch classifier is not designed to deal with so many classes, and so a way to determine what level of hierarchy will be used is needed. We propose to perform this choice with the help of a graph of similarity between classes (See Fig. 144 and 145

We first determinate how similar the classes are for the simple dimensionality descriptors, classifying with all the classes, and computing a class to class confusion matrix. This confusion matrix can be interpreted as an affinity between class matrix, and thus as a graph. We draw the graph using a spectral layout (team Networkx, 2014), which amounts to draw the graph following the first two eigen vector of the matrix (Similar to Principal Component Analysis). Those two vectors maximize the variance of the data (while being orthogonal), and thus best explain the data. This graph visually helps to choose the appropriate number of classes to use. A fully automatic method may be used via unsupervised clustering approach on the matrix (like The Affinity Propagation of Frey and Dueck, 2007).

Even when reducing the number of classes, the Paris dataset if unbalanced (some class have far less observations than some others). We tried two classical strategies to balance the data set regarding the number of observation per class. The first is under-sampling big classes : we randomly under-sample the observations to get roughly the same number of observation in every class.

The second strategy is to compute a statistical weight for every observation based on the class prevalence. This weight is then used in the learning process when building the Random Forest.

Using the confidence from the classifier

Contrary to classical classification method, we are not only interested in precision and recall per class, but also by the evolution of precision when prediction confidence varies.

In fact, for a filtering application, we can leverage the confidence information provided by the Random Forest method to artificially boost precision (at the cost of recall diminution). We can do this by limiting the minimal confidence allowed for every prediction. Orthogonaly, it is possible for some classes to increase recall at the cost of precision by using the result of a first patch classification and then incorporate in the result the other neighbour patches.

We stress that if the goal is to detect objects (and not classify each point), this strategy can be extremely efficient. For instance if we are looking for objects that are big enough to be in several patches (e.g. a car). In this case we can perform the classification (which is very fast and efficient), then keep only highly confident predictions, and then use the position of predictions to perform a local search for car limits. The classical alternative solution would be to perform a per point classification on each point, which would be extremely slow.

RESULT

Introduction to results

We design and execute several experiments in order to validate all points introduced in Section 7.3. First we prove that is it effectively possible to leverage points order, even using canonical open sources software out of the box. Second we perform MidOc ordering on very large point cloud and analyse the efficiency, quality and applications of the results. Third we use the number of points chosen in the MidOc ordering as a descriptors for a random forest classifier on two large data sets, proving their usefulness. Last we analyse the potential of this free descriptors, and what it brings when used in conjunction to other simple descriptors.

The base DBMS is team PostgreSQL, 2014. The spatial layer team PostGIS, 2014 is added to benefits from generic geometric types and multidimensional indexes. The specific point cloud storage and function come from pgPointCloud, 2014. The MidOc is either plpgsql or made in python with team SciPy, 2014. The classification is done with team Scikit, 2014, and the network clustering with team Networkx, 2014. Timings are only orders of magnitude due to the influence of database caching.

We use two data sets. There were chosen as different as possible to further evaluate how proposed methods can generalise on different data (See Figure fig:hist-density-dataset for histogram of patch density). The first data set is IQmulus, 2014 (Paris data set), an open source urban data set with varying density, singularities, and very challenging point cloud geometry. Every point is labelled with a hierarchy of 100 classes. The training set is only 12 millions points. Only 22 classes are represented. We group

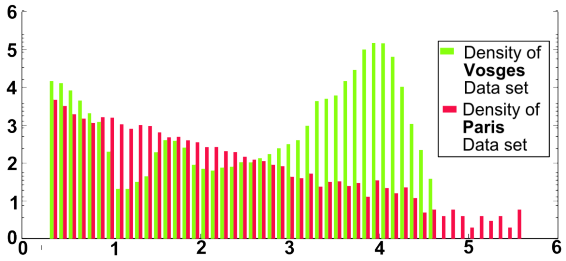


Figure 139: Histogram of number of points per patch, with a logarithmic scale for X and Y axis

points in $1m^3$ cubes. The histogram of density seems to follow an exponential law (See figure 139), the effect being that many patches with few points exist.

We also use the Vosges data set, which is a very wide spread, aerial Lidar, 5.5 Billions point cloud. Density is much more constant at 10k pts/patch . A vector ground truth about surface occupation nature (type of forest) is produced by the French Forest Agency. Again the classes are hierarchical, with 28 classes. We group points in 50 $\times$50m squares.

Using the Point Cloud Server for experiments

All the experiments are performed using a Point Cloud Server (cf Cura, 2014b). The key idea are that point clouds are stored inside a DBMS (postgres), as patch. Patch are compressed groups of points along with some basic statistics about points in the group. We hypothesize that in typical point cloud processing workflow, a point is never needed alone, but almost always with its surrounding points.

Each patch of points is then indexed in an R tree for most interesting attributes (obviously X,Y,Z but also time of acquisition, meta data, number of points, distance to source, etc.)

Having such a meta-type with powerful indexes allows use to find points based on various criteria extremely fast. (order of magnitude : ms). As an example, for a 2 Billion points dataset, we can find all patches in few milliseconds having : - between -1 and 3 meters high in reference to vehicle wheels - in a given 2D area defined by any polygon - acquired between 8h and 8h10 - etc.

The PCS offers an easy mean to perform data-partition based parallelism. We extensively use it in our experiments.

Exploiting the order of points

We proposed to implicitly store LOD in the order of the points (Section 7.3.2). In this first experiment we check that point cloud ordering is correctly preserved by common open source point cloud processing software. For this, we use a real point cloud, which we order by MidOc ordering. We export it as a text file as the reference file. For each software, we read the reference file and convert it into another format, then check that the

conversion did not change the order of points. The tree common open source software tested are CloudCompare[2] , LasTools[3] and Meshlab[4]. All pass the test.

MidOc: an ordering for gradual geometrical approximation

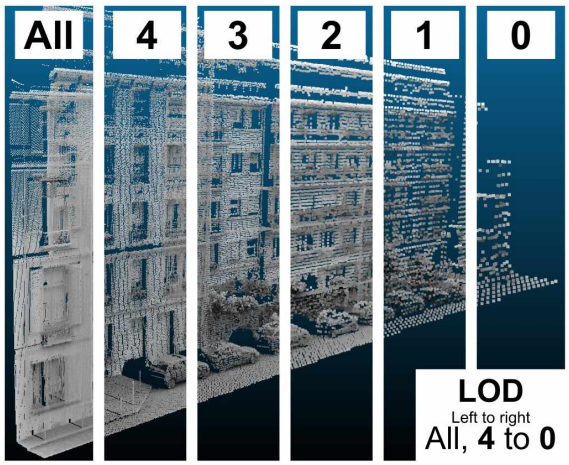


Figure 140: Schematic illustration of different LOD. Left to right, all points, then LOD 4 to 0. Visualized in cloud compare with ambient occlusion. Point size varies for better visual result.

We first test the visual fitness of MidOc ordering. Then we compute MidOc for our two datasets and evaluate the trade-off between point cloud size and point cloud LOD. As a proof of concept we stream a 3D point cloud with LOD to a browser.

The figure 140 illustrates LOD on a typical street of Paris dataset The figure 138 shows LOD on common street objects of various dimensionality.

We compute the size and canonical transfer time associated for a representative street point cloud. For this order of magnitude, the size is estimated at 5*4 Byte (5 floats) per point, and the (internet) transfer rate at 1 Mbyte/s.

Table 11: Number of points per LOD for the point cloud in the Figure 140 , plus estimated transfer time at 1 Mbyte/s.

Level	Typical spacing (cm)	Points number (k)	Percent of total size	Estimated time (s)
All	0.2 to 5	1600	100	60
0	100	3	0.2	0.1
1	50	11.6	0.7	0.4
2	25	41	2.6	1.5
3	12	134	8.4	5
4	6	372	23	14

We use 3 implementations of MidOc, two being pure plpgsql (postgreSQL script langage), and one Python (See Section 7.3.3.3). We successively order all the Paris and

2 www.danielgm.net/cc
3 www.cs.unc.edu/~isenburg/lastools
4 http://meshlab.sourceforge.net/

Vosges data sets with MidOc, using 20 parallel workers, with a plpgsql implementation. The ordering is successful on all patches, even in very challenging areas where there are big singularities in density, and many outliers. The total speed is about 100 millions points/hour using in-base processing. We prototyped an out-of-base processing where the extraction of points from patch is done on the client, and reached a 180 Mpts /h. The same method, without any ordering (only converting patch to point then point to patch) reach a 2.3 B pts/h. We consider it to be at least 10 times too slow for practical use. We briefly analyse performances, and conclude that only 10 workers are efficient.

As a proof of concept we stream points from the data base to a browser IGN, 2014. Because patch may contain a large number of points and because the browser WebGL technology is limited in how much points it can display, we limit the number of points per patch sent to the browser using LOD. Patch are ordered with MidOc, so the visual artifact is greatly reduced, and the data loads more quickly, as expected.

Excessive Density detection and correction

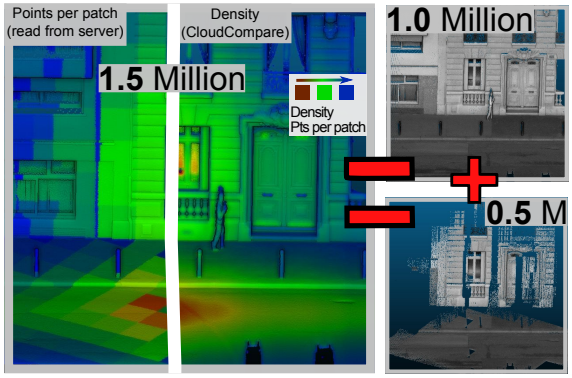


Figure 141: Abnormal density detection and correction. Top points per patch (left) or density (right), green-yellow-red. Bottom reflectance in grey.

We detect the abnormal density (explained in Section 7.3.5) in the Paris data set in ~ 100ms (See Figure 141). In comparison, computing the density per point with neighbourhood is extremely slow (only for this 1.5 Million extract, 1 minute with CloudCompare,4x2.5GHz, 10cm rad) (top right illustration), and after the density is computed for each points, all the point cloud still need to be filtered to find abnormal density spot.

If the patch are ordered following MidOc, unneeded points are removed by simply putting a threshold on points per patch (bottom left, 1 to 5k points /m^3, bottom right , 5k to 24 k pts /m^3). It considerably reduces the number of points (-33%).

This strategy can be automated by stating than no patch should return points over Level L_i. Then when getting points from the PCS, so that only points in those levels are sent.

Rough Dimensionality descriptor

We test the dimensionality descriptor (ppl) in two ways. First we compare the extracted (Dim_{LOD}) to the classical structure tensor based descriptor (Dim_{cov}). Second we assess

how useful it is for classification, by analysing how well it separates classes, and how much it is used when several other features are available.

Comparing LOD-based descriptor with Structure tensor-based descriptor

We compute Dim_{cov} following the indications of Weinmann et al., 2015 to get $p_{dim} -> [0..1]^3$, i.e. the probability to belong to [1D,2D,3D]. We convert this to Dim_{cov} with $Dim_{cov} = \sum_{i=1}^{3} i * p_{dim}[i]$.

Optionally, we test a filtering option so that the maximum distance in biggest two dimensions is more equivalent. However this approach fails to significantly improve results.

We test several method to extract Dim_{LOD} from ppl. The first method is to compute $Dim_{LODs}[i] = log2(ppl[i])/i$, which gives the simple dimensionality indice for each level. The second method is the same but work on occupancy evolution, with $Dim_{LODd}[i] = log2(ppl[i]/ppl[i-1])$ (discarding L_0). In both case the result is a dimensionality indice between 0 and 3 for each Level. We use both indices to fusion the dimensionality across Levels (working on $Dim_{LODA} = Dim_{LODs} \bigcup Dim_{LODd}$). The first method uses a RANSAC (team SciPy, 2014 implementation of Choi, Kim, and Yu, 2009) to find the best linear regression. The slope gives an idea of confidence (ideally, should be 0), and the value of the line at the middle of abscissa is an estimate of Dim_{LOD}. The second method robustly filters Dim_{LODA} based on median distance to median value and average the inliers to estimate Dim_{LOD}.

Dim_{cov} and Dim_{LOD} are computed with in-base and out-of-base processing, the latter being executed in parallel (8 workers). For 10k patches, 12 Mpts, retrieving data and writing result accounts for 48s, computing Dim_{LOD} to 8s, Dim_{cov} to 64s. Computing ppl (which is multiscale) using a linear octree takes between 58 L_6 and 85s L_8. (

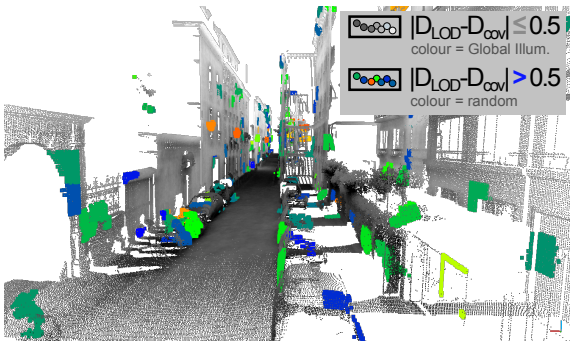


Figure 142: Dim_{LOD} and Dim_{cov} are mostly comparable, except for few patches (5%, coloured)

Comparing Dim_{cov} and Dim_{LOD} is not straightforward because the implicit definition of dimension is very different in the two methods. We analyse the patches where $|Dim_{LOD} - Dim_{cov}| <= 0.5$. 0.5 is an arbitrary threshold, but we feel that it represents the point above which descriptors will predict unreconcilable dimensions. Those patches represent 93% of the data set (0.96 % of points), with a correlation of 0.80. Overall the proposed dimensions are similar for the majority of patch, especially for well filled 1D and 2D patches (See Fig. 142).

We analyse the 684 remaining patches to look for possible explanations of the difference in dimension (See Fig. 143).

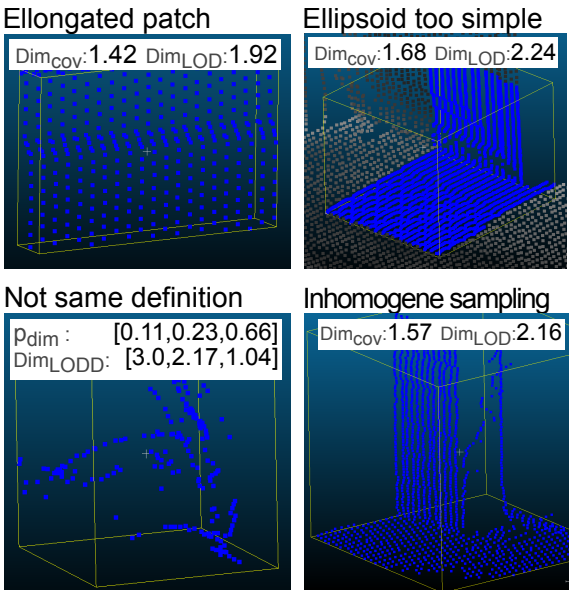


Figure 143: Representative patches for $|\mathrm{Dim}_{LOD}\text{-}\mathrm{Dim}_{cov} > 0.5|$. Most differences are explained by Dim_{cov} limitations (See 7.4.6.1).

We consider the following four main sources of limitations from Dim_{cov}.

- Elongated patch.
  Dim_{cov}=1.42 ,Dim_{LOD}=1.92. If the patch is not roughly a square,Dim_{cov} gives a bad estimation as it is biased by the un-symmetry of point distribution.

- Ellipsoid too simple.
  Dim_{cov}=1.68, Dim_{LOD}=2.24. Dim_{cov} fits an ellipsoid, which can not cope with complex objects, especially when the barycentre does not happen to be at a favourable place.

- Coping with heterogeneous sampling.
  p_{dim}=[0.56,0.32,0.12], Dim_{cov}=1.57, Dim_{LOD}=2.16.
  Dim_{cov} is sensitive to difference in point density. The points on the bottom plan are much 3 times less dense than in the vertical plan, leading to a wrong estimate.

- Definition of dimension different.
  General:$\mathrm{Dim}_{cov} \in [1.2, 2.6]$, $\mathrm{Dim}_{LOD} \in [1.7..2.7]$
  This patch: p_{dim}=[0.11, 0.23, 0.66]
  ppl=[1,8,36,74..], Dim_{LODD}=[3.0,2.17,1.04]. Trees are a good example of how the two descriptors rely on a different dimension definition. For Dim_{cov} points may be well spread out, so usually p_{3D} is high. Yet, tree patches are also subject to density variation, and may also be elongate, which renders Dim_{cov} very variable. On the opposite, Dim_{LOD} considers the dimensionality at different scale (See Fig. 153). From afar a tree-patch is volumetric, at lower scal, it seems planar (leaf and small sticks form rough plans together). Lastly at small scale, the tree looks linear (sticks).

Usefulness of rough descriptor for classification

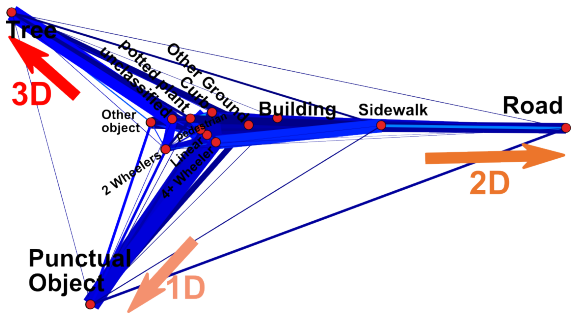


Figure 144: Spectral clustering of confusion matrix of Paris data set classification using only ppl descriptor. Edge width and colour are proportional to affinity. Node position is determined fully automatically. Red-ish arrows are manually placed as visual clues.

Using only the ppl descriptor, a classification is performed on Paris data set, then a confusion matrix is computed. We use the spectral layout (see Section 7.3.6.2) to automatically produce the graph in Figure 144. We manually ad 1D,2D and 3D arrows. On this graph, classes that are most similar according to the classification with ppl are close. The graph clearly present an organisation following 3 axis. Those axis coincide with the dimensionality of the classes. For instance the "tree" classe as a strong 3D dimensionality. The "Punctual object" class, defined by "Objects which representation on a map should be a point", is strongly 1D (lines), with object like bollard and public light. The "Road" class is strongly 2D, the road being locally roughly a plan. The center of the graph is occupied by classes that may have mixed dimensionality, for instance "4+ wheeler" (i.e. car) may be a plan or more volumetric, because of the $1m^3$ sampling. "Building" and "sidewalk" are not as clearly 2D as the "road" class. Indeed, the patch of "sidewalk" class are strongly mixed (containing 22 %of non-sidewalk points, See figure 147). The building class is also not pure 2D because building facade in Paris contains balcony, building decoration and floors, whcih introduce lots of object-like patches, which explain that building is closer to the object cluster. (See Figure 140, in the Level 3 for instance). The dimensionality descriptor clearly separates classes following their dimensionality, but can't separate classes with mixed dimensionality.

To further evaluate the dimensionality descriptor, we introduce other classification features (see 7.4.7), perform classification and compute each feature importance. The overall precision and recall result of these classification is correct, and the ppl descriptor is of significant use (See Figure 147 and 146), especially in the Vosges data set. The ppl descriptor is less used in Paris data set, maybe because lots of classes can not really be defined geometrically, but more with the context.

Patch Classification

Introducing other features

The dimensionality descriptor alone cannot be used to perform sophisticated classification, because many semantically different objects have similar dimension (for instance, a piece of wall and of ground are dimensionally very similar , yet semantically very dif-

ferent). We introduce additional simple features for classification (See Section 7.3.6.2). All use already stored patch statistics, and thus are extremely fast to compute. (P : for Paris , V : for Vosges: - average of altitude regarding sensing device origin(P) - area of patch_bounding_box (P) : - patch height (P) - points_per_level (ppl), level 1 to 4 (P+V) - average of intensity (P+V) - average of number_of_echo (P+V) - average Z (V)

For Vosges data set, we reach a speed of 1 Mpoints/s/worker to extract those features.

Classification Setting

Undersampling and weighting are used on the paris dataset. First Undersampling to reduce the over dominant building classe to a 100 factor of the smallest class support. Then weighting is used to compensate for differences in support. For the Vosges data set only the weighting strategy is used. The weighting approach is favoured over undersampling because it lessen variability of results when classes are very heterogeneous.

To ensure significant results we follow a K-fold cross-validation method. We again compute a confusion matrix (i.e. affinity between classes) on the Paris data set to choose which level of class hierarchy should be used. fig:class-clustering-all-features

Analysing class hierarchy

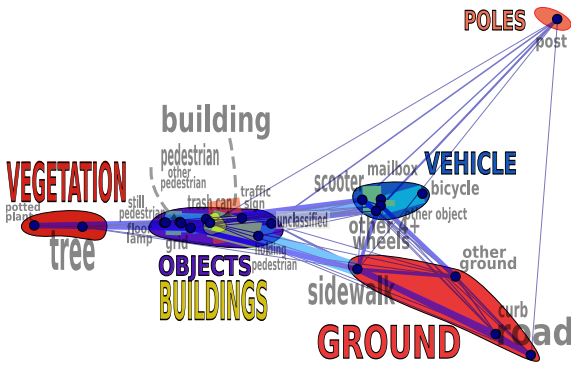


Figure 145: Result of automatic spectral clustering over confusion matrix for patch classification of Paris data set with all simple features. Edges width and colour are proportional to confusion. Manually drawn clusters for easier understanding.

Choosing which level of the class hierarchy to use depends on data set and applications. In a canonical classification perspective, we have to strongly reduce the number of classes if we want to have significant results. However reducing the number of class (i.e use a higher level in the classes hierarchy) also means that classes are more heterogeneous.

Both data set are extremely unbalanced (factor 100 or more). Thus our simple and direct Random Forest approach is ill suited for dealing with extremely small classes. (Cascading or one versus all framework would be needed).

For Vosges data set a short analysis convince us to use 3 classes: Forest, Land, and other, on this three classes, the Land class is statistically overweighted by the 2 others.

For the Paris data set, we again use a spectral layout to represent the affinity graph (See Figure 145). Introducing other features clearly helps to separate more classes. The graph also shows the limit of the classification, because some class cannot be properly

defined without context (e.g. the side-walk, which is by definition the space between building and road, hence is defined contextually).

Classification results

	prec.	rec.	supp.	mix.
Closed Forest	**0.99**	**0.91**	**390k**	**0.883**
Moor	0.18	0.68	8.7k	0.741
Not forest	**0.86**	**0.89**	**128k**	**1**
avg/total	0.94	0.90	526k	0.901

Feature usage

ppl_1	ppl_2	ppl_3	ppl_4	Intensity	nber of echo	mean Z	patch height
0.57 (0.07+0.13+0.18+0.19)				0.09	0.06	**0.12**	**0.14**

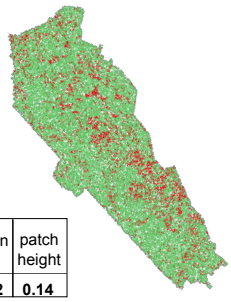


Figure 146: Vosges dataset. (table 2) Precision(prec.), recall (rec.), support (supp.), and average percent of points of the class in the patches, for comparison with point based method (mix.). (table 1)Feature usage

We perform a analysis of error on Vosges dataset and we note that errors seem to be significantly correlated to distance to borders.

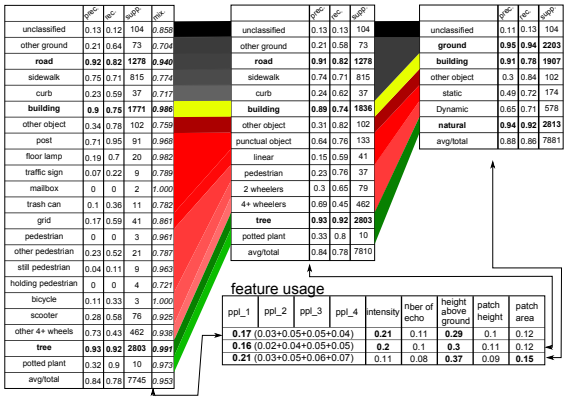


Figure 147: Results for Paris data set: at various level of class hierarchy. Precision(prec.), recall (rec.), support (sup.) and average percent of points of the class in the patches of the class, for comparison with point based method (mix.). Classes of the same type are in the same continuous tone. Feature usage is evaluated for each level in the class hierarchy.

The learning time is less than a minute, the predicting time is less than a second.

For both dataset, patches main contain points from several classes. We measure how much of the patch points pertain to the dominant class. The result is given in the columns "mix". For instance the patch of the class "building" contains an average of 98.6 %points of the class "building", whereas the patch from the class "forest" contains 88.3 %points of the class "forest". Therefore, to provide a comparison with point based classification, we can compute the precision of the classification per point as $\text{Precision}_{\text{point}} = \text{Precision}_{\text{patch}} * \text{Mix}$.. (Same for recall).

Precision or Recall increase

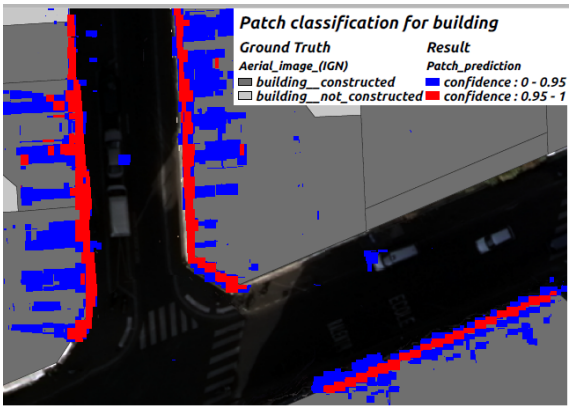


Figure 148: Plotting of patches classified as building, using confidence to increase precision. Ground truth from IGN and Open Data Paris

As explained in Section 7.3.6.3, we can leverage the random forest confidence score to artificially increase the precision.

We focus on the building class. As seen in the Figure 148, initial classification results (blue) are mostly correct. Yet, only keeping patches with high confidence may greatly increase precision (to 100 %). Further filtering on confidence can not increase precision, but will reduce the variability of the found building patches. This result (red) would provides a much better base for building reconstruction for instance.

The patch classifier can also be used as a filtering preprocess. In this case, the goal is not to have a great precision, but to be fast and with a good recall. Such recall may be increased artificially for class of objects bigger than the sampling size ($1m^3$ for Paris).

We take the example of ground classification (See Figure 149). The goal is to find all ground patches very fast. We focus on a small area for illustration purpose. This area contains 3086 patches, including 439 ground patches. Patch classification finds 421 ground patch, with a recall of 92.7%. Using the found patch, all the surrounding patches (X,Y : 2 m, Z : 0.5 m) are added to the result (few seconds). There are now 652 patches in the result, and the recall is 100%. This means that from a filtering point of view, a complex classifier that would try to find ground points can be used on $652/3086 = 21\%$ of the data set, at the price of few seconds of computing, without any loss of information.

DISCUSSION

Point cloud server

We refer the reader to Cura, Perret, and Paparoditis, 2015b for an exhaustive analyse of the Point Cloud Server. Briefly, the PCS has demonstrated all the required capacitites to manage point clouds and scale well. To the best of our knowledge the fastest and easiest way to filter very big point cloud using complex spatial and temporal criteria, as well as natively integrate point cloud with other GIS data (raster, vector). The main limitation is based on the hypothesis than points can be regrouped into meaningful (regarding

Figure 149: Map of patch clustering result for ground. The classical result finds few extra patches that are not ground (blue), and misses some ground patches (red). Recall is increased by adding to the ground class all patches that are less than 2 meters in X,Y and 0.5 meter in Z around the found patches. Extra patches are much more numerous, but all the ground patches are found.

further point utilisation) patches. If this hypothesis is false, the PCS lose most of its interest.

Exploiting the order of points

From a practical point of view, implicitly storing the LOD using the point ordering seems to be estremely portable. Most softwares would not change the order of points. For those who might change the order of points, it is still very easy to add the order as an attribute, thus making it fully portable. However, this approach has two limitations. The first limitation is that the order of point might already contains precious information. For instance with a static Lidar device, the acquisition order allows to reconstruct neighbourhood information. The second limitation is that an LOD ordering might conflict with compression. Indeed ordering the points to form LOD will create a list of points were successive points are very different. Yet compressing works by exploiting similarities. A software like LasTool using delta compressing might suffer heavily from this.

We stress that the LOD are in fact almost continuous (as in the third illustrations of Fig. 130).

MidOc is a way to order points based on their importance. In MidOc, the importance is defined geometrically. Yet specific applications may benefit from using other measure of importance, possibly using other attributes than geometrical one, and possibly using more perceptual-oriented measures.

MidOc relies on two hypothesis which might be false in some case. Indeed, variation of density may be a wanted feature (e.g. stereovision, with more image on more important parts of the object being sense). Again the low geometrical noise hypothesis might be true for Lidar, but not for Stereo vision or medical imaging point cloud. However in those case denoising methods may be applied before computing MidOc.

Applications

MidOc ordering might be of use in 3 types of applications. First it can be used for graphical LOD, as a service for point cloud visualisation. Second the ordering allows to correct density to be much more constant. Complex processing methods may benefits from an almost constant density, or for the absence of strong density variation. Third the ordering can be used for point cloud generalisation, as a service for processing methods that may only be able to deal with a fraction of the points.

The illustration 140 gives visual example of LOD result and how it could be used to vary density depending on the distance to camera. Figure 138 also gives visual examples for common objects of different dimensionality. It is visually clear that the rate of increase of points from LOD 0 to 4 for floor lamp (1D) window (2D) and tree (3D) is very different. Small details are also preserved like the poles or the antenna of the car. preserving those detail with random or distance based subsampling would be difficult.

Implementation

Octree construction may be avoided by simply reading coordinates bitwise in a correctly centred/scaled point cloud. We centre a point cloud so that the lowest point of all dimension is $(0,0,0)$, and scale it so that the biggest dimension is in $[0,1[$. The point cloud is then quantized into $[0..2**L-1]$ for each coordinate. The coordinates are now integers, and for each point, reading its coordinates bitwise left to right gives the position of the point in the octree for level of the bit read. This means performing this centring/scaling/quantization directly gives the octree. Moreover, further operations can be performed using bit arithmetic, which is extremely fast.

On this illustration the point P has coordinates $(5,2)$ in a $[0,2^3-1]^2$ system. Reading the coordinates as binary gives $(b'101', b'010')$. Thus we know that on the first level of a quad tree, P will be in the right (x=b'1xx') bottom (y=b'0yy') cell. For the next level, we divide the previous cell in 2, and read the next binary coordinate. P will be in the left (x=b'x0x') up (y=b'y1y') cell. There is no computing required, only bit mask and data reading.

Regarding implementation, the three we propose are much too slow, by an order of magnitude to be easily used in real situation. We stress however that the slowness comes from ineficient data manipulation, rather than from the complexity of the ordering. It

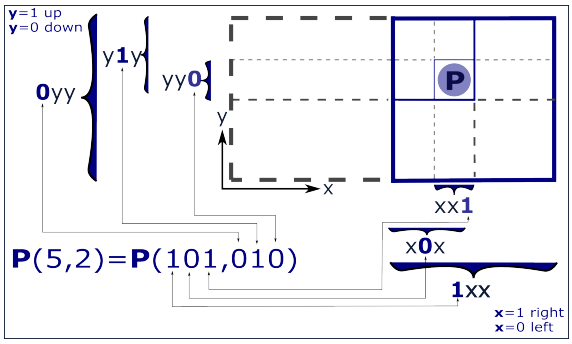


Figure 150: Principle of binary coordinates for a centered, scaled and quantized point cloud.

may also be possible to use the revert Hilbert ordering to directly compute MidOc. Furthermore, octree construction has been commonly done on GPU for more than a decade.

Size versus LOD trade-off

The table 11 shows that using the level 3 may speed the transfer time by a 10 factor. The point cloud server throughput is about 2-3 Mbyte /s(monoprocess), sufficient for an internet troughput, but not fast enough for a LAN 10 Mbyte /s. This relatively slow troughput is due to current point cloud server limitation (cf 7.5.1).

Large scale computing

The relatively slow computing (180 Millions points /h) is a very strong limitation. This could be avoided. A C implementation which can access raw patch would also be faster for ordering points.

LOD stream

Streaming low level of detail patches greatly accelerate visualisation, which is very useful when the full point cloud is not needed. To further accelerate transmission, patch LOD can be determined according to the distance to camera (frustrum culling). (See Figure 151 for a naive visual explanation.)
As seen before (See Section 7.5.3.3), the point cloud server is fast enough for an internet connection, but is currently slower than a file-based points streaming. Thus for the moment LOD stream is interesting only when bandwidth is limited.

Excessive Density detection and correction

Fast detection

Density abnormality detection at the patch level offer the great advantage of avoiding to read points. This is the key to the speed of this method. We don't know any method that is as fast and simple.
The limitations stems from the aggregated nature of patch. the number of points per

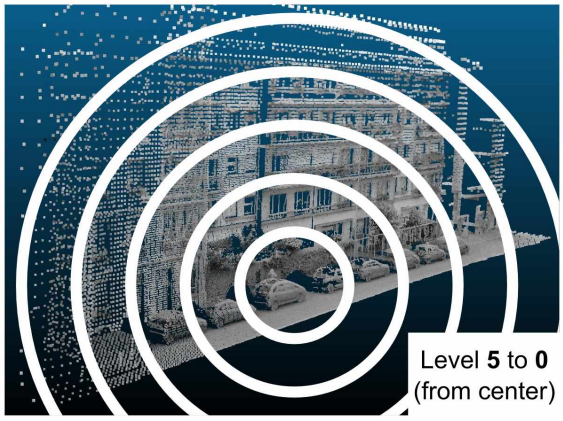


Figure 151: Schematique example of LOD depending on distance to camera

patch doesn't give the density per point, but a quantized version of this per patch. So it is not possible to have a fine per point density.

Simple correction

The correction of density peak we propose has the advantage of being instantaneous and not induce any data loss. It is also easy to use as safeguard for an application that may be sensible to density peak : the application simply defines the highest number of points /m³it can handle, and the Point cloud server will always output less than that. The most important limitation this method doesn't guarantee homogeneous density, only a maximum density. For instance if an application requires 1000 points /m³for ground patches, all the patches must have more than 1000 points, and patch must have been ordered with MidOc for level 0 to at least 5 ($4^5 = 1024$). The homogeneous density may also be compromised when the patch are not only split spatially, but with other logics (in our case, points in patch can not be separated by more than 30 seconds, and all points in a patch must come from the same original acquisition file).

Crude dimensionality descriptor (MidOc by-product)

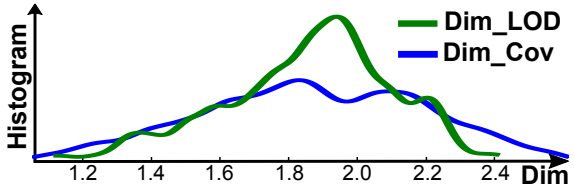


Figure 152: Histogram of $\mathtt{Dim}_{\mathtt{LOD}}$ and $\mathtt{Dim}_{\mathtt{cov}}$ for patch in trees (500 kpts). Tree dimension could be from 1.2 to 2.6, yet $\mathtt{Dim}_{\mathtt{LOD}}$ is less ambiguous than $\mathtt{Dim}_{\mathtt{cov}}$

Tree patches are challenging for both dimensionality descriptor. There possible dimension changes a lot (See Fig. 152), although $\mathtt{Dim}_{\mathtt{LOD}}$ is more concentrated. Yet, ppl is extremely useful to classify trees. Indeed, ppl contains the dimensionality at various scale, and potentially the variation of it, which is quite specific for trees (See Fig. 153).

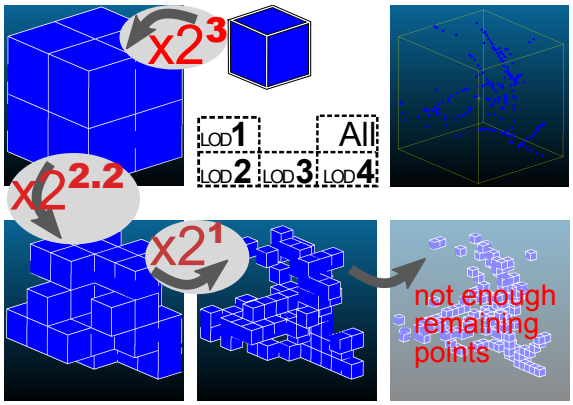


Figure 153: Evolution of tree patch octree cells occupancy, illustrating the various dimensions of trees depending on the scale. (Dimension is embeded it the power of 2).

We stress that a true octree cell occupancy (i.e. without picking points as in the ppl) can be obtained without computing the octree, simply by using the binary coordinates (See 150). We implement it in python as a proof of concept. Computing it is about as fast as computing $\mathtt{Dim}_{\mathtt{cov}}$.

Overall, ppl offers a good alternative to the classical dimensionality descriptor ($\mathtt{Dim}_{\mathtt{cov}}$), being more robust and multiscale. However the ppl also has limitations. First the quality of the dimensionality description may be affected by a low number of points in the patch. Second in some case it is hard to reduce it to a meaningful $\mathtt{Dim}_{\mathtt{LOD}}]$. Last because of assumption on density, it is sensible to geometric noise.

Patch Classification

The ppl descriptor contains lots of information about the patch. This information is leveraged by the Random Forest method and permit a rough classification based on geometric differences. As expected, ppl descriptor are not sufficient to correctly separate complex objects, which is the main limitation for a classification application.

The additional features are extremely simple, and far from the one used in state of the art. Notably, we don't use any contextual feature. We choose to classify directly in N classes, whereas due to the large unbalance indataset, cacade or 1 versus all approaches would be more adapted.

Analysing class hierarchy

The figure 145 shows the limit of a classification without contextual information. For instance the class grid and buildings are similar because in Paris buildings balcony are typically made of grids.

To better identify confusion between classes, we use a spectral layout on the affinity matrix. Graphing this matrix in 2D ammount to a problem of dimensionality reduction. It could use more advanced method than simply using the first two eigen vector, in perticular the two vector wouldn't need to be orthogonal (for instance, like in Hyvärinen and Oja, 2000).

Classification results

First the feature usage for vosges data set clearly shows that amongst all the simple descriptor, the ppl descriptor is largely favored. This may be explained by the fact that forest and bare land have very different dimensionality, which is conveyed by the ppl descriptor.

Second the patch classifier appears to have very good result to predict if a patch is forest or not. The precision is almost perfect for forest. We reach the limit of precision of ground truth. Because most of the errors are on border area, the recall for forest can also be easily artificially increased. The percent of points in patch that are in the patch class allow to compare result with a point based method. For instance the average precision per point for closed forest would be $0.99 * 0.883 = 0.874$. We stress that this is averaged results, and better precision per point could be achieved because we may use random forest confidence to guess border area (with a separate learning for instance). For comparison with point methods, the patch classifier predict with very good precision and support over 6 billions points in few seconds (few minutes for training). We don't know other method that have similar result while being as fast and natively 3D. The Moor class can't be separated without more specialised descriptor, because Moor and no forest classes are geometrically very similar.

The principal limitation is that for this kind of aerial Lidar data set the 2.5D approximation may be sufficient, which enables many raster based methods that may perform better or faster.

The figure 147 gives full results for paris data set, at various class hierarchy level. Because the goal is filtering and not pure classification, we only comment the 7 classes result. The proposed methods appears very effective to find building, ground and trees. Even taking into consideration the fact that patch may contains mixed classes (column mix.), the result are in the range of state of the art point classifier, while being extremely fast. This result are sufficient to increase recall or precision to 1 if necessary. We stress that even results appearing less good (4+wheelers , 0.69 precision, 0.45 recall) are in fact sufficient to increase recall to 1 (by spatial dilatation of the result), which enables then to use more subtle methods on filtered patches.

ppl descriptor is less used than for the Vosges data set, but is still useful, particularly when there are few classes. It is interesting to note that the mean intensity descriptor seems to be used to distinguish between objects, which makes it less useful in the 7 classes case. The patch classifier for Paris data set is clearly limited to separate simple classes. In particular, the performances for objects are clearly lower than the state of the art. A dedicated approach should be used (cascaded or one versus all classifier).

Estimating the speed and performance of patch based classification compared to point based classification

The Point Cloud Server is designed to work on patches, which in turns enable massive scaling.

Timing a server is difficult because of different layer of caches, and background workers. Therefore, timing should be considered as order of magnitude. For Paris data set,extracting extra classification features requires $\sim \frac{400s}{n_{workers}}$(1 to 8 workers), learning $\sim$ 210s, and classification $\sim$ few s. We refer to Weinmann et al., 2015(Table 5) for point classification timing on the same dataset (4.28h, 2s, 90s) (please note that the training

set is much reduced by data balancing). As expected the speed gain is high for complex feature computing (not required) and point classification (done on patch and not points in our case).

For Vosges data set, features are extracted at 1Mpts/s/worker, learning ~ fewmin, classification ~ 10s. The Vosges data set has not been used in other articles, therefore we propose to compare the timings to Shapovalov, Velizhev, and Barinova, 2010 (Table 3). Keeping only feature computation and random forest training (again on a reduced data set), they process 2 Mpoints in 2 min, whereas our method process the equivalent of 5.5 B points in few minutes.

Learning and classification are monothreaded (3 folds validation), although the latter is easy to parallelise. Overall, the proposed method is one to three orders of magnitude faster.

For Paris data set (Fig. 147), we compare to Weinmann et al., 2015(Table 5). As expected there results are better, particularly in terms of precision (except for the class of vegetation). This trend is aggravated by taking into account the "mix." factor. Indeed we perform patch classification, and patch may not pertain to only one class, which is measured by the mix factor (amount of points in the main class divided by the total number of point). However, including the mix factor the results are still within the 85 to 90 % precision for the main classes (ground, building, natural).

For Vosges data set (Fig 146), we refer to Shapovalov, Velizhev, and Barinova, 2010 (Table 2). There random forest classifier get trees with 93% precision and 89% recall. Including the mix factor we get trees with a precision of 87% and 80% recall. As a comparison to image based classification, an informal experiment of classification with satellite image reaches between 85 % and 93 % of precision for forest class depending on the pixel size (between 5 and 0.5 m).

Overall, the proposed method get reasonably good results compared to more sophisticated methods, while being much faster. It so makes a good candidate as a preprocessing filtering step.

Precision or Recall increase

Because the propose methods are preprocess of filtering step, it can be advantageous to increase precision or recall.

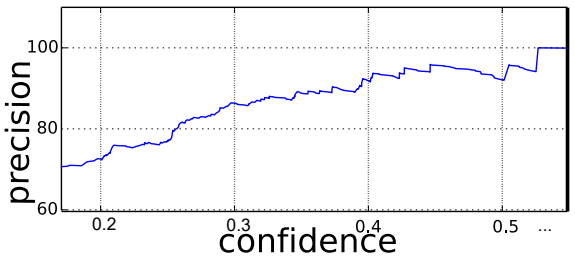


Figure 154: Precision of 4+wheelers class is a roughly rising function of random forest confidence scores.

In the Figure 148 gives a visual example where increasing precision and reducing class heterogeneity is advantageous. This illustrates that having a 1 precision or recall is not necessary the ultimate goal. In this case it would be much easier to perform line

detection starting from red patches rather than blue patches.
The limitation of artificial precision increase is that it is only possible when precision is roughly a rising function of random forest confidence, as seen on the illustration 154. For this class, by accepting only prediction of random forest that have a confidence over 0.3 the precision goes from 0.68 to 0.86, at the cost of ignoring half the predictions for this class. This method necessitates that the precision is roughly a rising function of the confidence, as for the 4+wheeler class for instance (See Figure 154). This strategy is not possible for incoherent class, like unclassified class.

The method we present for artificial recall increase is only possible if at least one patch of each object is retrieved, and objects are spatially dense. This is because a spatial dilatation operation is used. This is the case for "4+wheelers" objects in the paris data set for instance. The whole method is possible because spatial dilatation is very fast in point cloud server (because of index). Moreover, because the global goal is to find all patches of a class while leaving out some patches, it would be senseless to dilate with a very big distance. In such case recall would be 1, but all patches would be in the result, thus there would be no filtering, and no speeding.
The limitation is that this recall increase method is more like a deformation of already correct results rather than a magical method that will work with all classes.

CONCLUSION

Using the Point Cloud Server, we propose a new paradigm by separating the spatial indexing and LOD scheme. Subdivision of point clouds into groups of points (patches) allows us to implicitly store LOD into the order of points rather than externally. After an ordering step, exploiting this LOD does not require any further computation. We propose an geometrical ordering (MidOc) based on the closest point to octree cell centre that produces reliable LOD, successfully used for visualization or as a service for other processing methods (density correction/reduction). By also performing intra-level dedicated ordering, we create LOD that can be used partially and still provide good coverage. Furthermore, by collecting the number of points per octree level, an information available during MidOc ordering, we create a multi-scale dimensionality descriptor. We show the interest of this descriptor, both by comparison to the state of the art and by proof of its usefulness in real Lidar dataset classification. Classification is extremely fast, sometime at the price of performance (precision / recall). However we prove that those results can be used as a pre-processing step for more complex methods, using if necessary precision-increase or recall-increase strategies.

BIBLIOGRAPHY

Agarwal, Sameer, Keir Mierle, and Others (2016). "Ceres Solver." In:

Ahmed, Mahmuda, Sophia Karagiorgou, Dieter Pfoser, and Carola Wenk (2014). "A comparison and evaluation of map construction algorithms using vehicle tracking data." en. In: *Geoinformatica* 19.3, pp. 601–632. ISSN: 1384-6175, 1573-7624. DOI: 10.1007/s10707-014-0222-6.

Aichholzer, Oswin, Franz Aurenhammer, David Alberts, and Bernd Gärtner (1996). "A Novel Type of Skeleton for Polygons." en. In: *J.UCS The Journal of Universal Computer Science*. Ed. by Hermann Maurer, Cristian Calude, and Arto Salomaa. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 752–761. ISBN: 978-3-642-80352-9 978-3-642-80350-5.

Amit, Yali and Donald Geman (1997). "Shape quantization and recognition with randomized trees." In: *Neural Computation* 9, pp. 1545–1588.

Applegate, C. S., S. D. Laycock, and A. M. Day (2011). "A sketch-based system for highway design." In: *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*. SBIM '11. New York, NY, USA: ACM, pp. 55–62. ISBN: 978-1-4503-0906-6. DOI: 10.1145/2021164.2021175.

ArchiExpo (2014). *ArchiExpo - Public spaces*. http://www.archiexpo.com/cat/public-spaces-0.html.

Aubrecht, C., K. Steinnocher, M. Hollaus, and W. Wagner (2009). "Integrating earth observation and GIScience for high resolution spatial and functional modeling of urban land use." In: *Computers, Environment and Urban Systems* 33.1, pp. 15–25. ISSN: 0198-9715. DOI: 10.1016/j.compenvurbsys.2008.09.007.

Azim, A. and O. Aycard (2012). "Detection, classification and tracking of moving objects in a 3D environment." In: *2012 IEEE Intelligent Vehicles Symposium (IV)*, pp. 802–807. DOI: 10.1109/IVS.2012.6232303.

Baert, Jeroen, Ares Lagae, and Ph Dutré (2014). "Out-of-Core Construction of Sparse Voxel Octrees." In: *Computer Graphics Forum*. Vol. 33. Wiley Online Library, pp. 220–227.

Bar Hillel, Aharon, Ronen Lerner, Dan Levi, and Guy Raz (2012). "Recent progress in road and lane detection: a survey." In: *Machine Vision and Applications*. ISSN: 0932-8092, 1432-1769. DOI: 10.1007/s00138-011-0404-2.

Baraniuk, Richard, Mark Davenport, Marco Duarte, chinmay Hegde, Jason Laska, Mona Sheikh, and Wotao Yin (2011). "An Introduction to Compressive Sensing." anglais. In: Connexions. Houston, Texas: Rice University, p. 118.

Beneš, Bedřich, Michel A. Massih, Philip Jarvis, Daniel G. Aliaga, and Carlos A. Vanegas (2011). "Urban ecosystem design." In: *Symposium on Interactive 3D Graphics and Games*. I3D '11. New York, NY, USA: ACM, pp. 167–174. ISBN: 978-1-4503-0565-5. DOI: 10.1145/1944745.1944773.

Beneš, Jan, Alexander Wilkie, and Jaroslav Křivánek (2014). "Procedural Modelling of Urban Road Networks." en. In: *Computer Graphics Forum* 33.6, pp. 132–142. ISSN: 1467-8659. DOI: 10.1111/cgf.12283.

Bereuter, Pia (2015). "Quadtree-based Real-time Point Generalisation for Web and Mobile Mapping." English. PhD thesis. Zurich: Mathematisch-naturwissenschaftlichen Fakultät der Universität Zürich.

Bertails-Descoubes, Florence (2012). "Super-Clothoids." In: *Comp. Graph. Forum* 31.2pt2, pp. 509–518. ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.03030.x.

Besl, Paul J. and Neil D. McKay (1992). "Method for registration of 3-D shapes." In: ed. by Paul S. Schenker, pp. 586–606. DOI: 10.1117/12.57955.

Bessmeltsev, Mikhail, Caoyu Wang, Alla Sheffer, and Karan Singh (2012). "Design-driven quadrangulation of closed 3D curves." In: *ACM Trans. Graph.* 31.6, 178:1–178:11. ISSN: 0730-0301. DOI: 10.1145/2366145.2366197.

Bier, Eric A., Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose (1993). "Toolglass and magic lenses: the see-through interface." In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. ACM, pp. 73–80.

Bloomenthal, J. (1985). "Modeling the mighty maple." In: *ACM SIGGRAPH Computer Graphics*. Vol. 19, pp. 305–311.

Bokeloh, M., M. Wand, V. Koltun, and H. P Seidel (2011). "Pattern-aware shape deformation using sliding dockers." In: *ACM Transactions on Graphics (TOG)* 30.6, p. 123.

Bokeloh, Martin, Michael Wand, Hans-Peter Seidel, and Vladlen Koltun (2012). "An algebraic model for parameterized shape editing." In: *ACM Trans. Graph.* 31.4, 78:1–78:10. ISSN: 0730-0301. DOI: 10.1145/2185520.2185574.

Boyko, Aleksey and Thomas Funkhouser (2011). "Extracting roads from dense point clouds in large scale urban environment." en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 66.6, S2–S12. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2011.09.009.

Breiman, Leo (2001). "Random Forests." In: *Machine Learning*, pp. 5–32.

Brenner, Claus (2010). "Vehicle localization using landmarks obtained by a lidar mobile mapping system." In: *Proceedings of the Photogrammetric Computer Vision and Image Analysis* 38, pp. 139–144.

Bruneton, Éric and Fabrice Neyret (2012). "Real-time Realistic Rendering and Lighting of Forests." In: *Comput. Graph. Forum*.

Burghardt, Dirk, Cécile Duchêne, and William Mackaness, eds. (2014). *Abstracting Geographic Information in a Data Rich World*. Lecture Notes in Geoinformation and Cartography. Cham: Springer International Publishing. ISBN: 978-3-319-00202-6 978-3-319-00203-3.

Bustos, Benjamin, Daniel A. Keim, Dietmar Saupe, Tobias Schreck, and Dejan V. Vranić (2005). "Feature-based similarity search in 3D object databases." en. In: *ACM Computing Surveys* 37.4, pp. 345–387. ISSN: 03600300. DOI: 10.1145/1118890.1118893.

Cabral, M., S. Lefebvre, C. Dachsbacher, and G. Drettakis (2009). "Structure-Preserving Reshape for Textured Architectural Scenes." In: *Computer Graphics Forum*. Vol. 28, pp. 469–480.

Chaudhuri, S., E. Kalogerakis, L. Guibas, and V. Koltun (2011). "Probabilistic reasoning for assembly-based 3D modeling." In: *ACM Transactions on Graphics (TOG)*. Vol. 30, p. 35.

Chen, Guoning, Gregory Esch, Peter Wonka, Pascal Müller, and Eugene Zhang (2008). "Interactive procedural street modeling." In: *ACM Transactions on Graphics* 27.3, Article 103: 1–10. ISSN: 07300301. DOI: 10.1145/1360612.1360702.

Choi, S., T. Kim, and W. Yu (2009). "Performance evaluation of ransac family." In: *Proceedings of the British Machine Vision Conference*.

Chu He, Fang Yang, Sha Yin, Xinping Deng, and Mingsheng Liao (2013). "Stereoscopic Road Network Extraction by Decision-Level Fusion of Optical and SAR Imagery." In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 6.5, pp. 2221–2228. ISSN: 1939-1404, 2151-1535. DOI: 10.1109/JSTARS.2013.2249656. (Visited on 05/12/2016).

Chum, Ondrej and Jirı Matas (2002). "Randomized RANSAC with Td, d test." In: *Proc. British Machine Vision Conference*. Vol. 2, pp. 448–457.

Clementini, Eliseo and Robert Laurini (2008). "Un cadre conceptuel pour modéliser les relations spatiales." In: *Revue des Nouvelles Technologies de l'Information (RNTI)* 14, pp. 1–17.

Clode, Simon, Franz Rottensteiner, Peter Kootsookos, and Emanuel Zelniker (2007). "Detection and vectorization of roads from lidar data." In: *Photogrammetric Engineering & Remote Sensing* 73.5, pp. 517–535.

Cornelis, N., B. Leibe, K. Cornelis, and L. Van Gool (2008). "3d urban scene modeling integrating recognition and reconstruction." In: *International Journal of Computer Vision* 78.2, pp. 121–141.

Coughlan, James M. and Alan L. Yuille (1999). "Manhattan world: Compass direction from a single image by bayesian inference." In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. Vol. 2. IEEE, pp. 941–947.

Cullen, B. and C. O'Sullivan (2011). "A caching approach to real-time procedural generation of cities from GIS data." In: *Journal of WSCG* 19.3, pp. 119–126.

Cura, R., J. Perret, and N. Paparoditis (2015a). "STREETGEN: IN-BASE PROCEDURAL-BASED ROAD GENERATION." en. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* II-3/W5, pp. 409–416. ISSN: 2194-9050. DOI: 10.5194/isprsannals-II-3-W5-409-2015.

Cura, Rémi (2014a). *A PostgreSQL Server for Point Cloud Storage and Processing*. Paris.

— (2014b). *A postgres server for point clouds storage and processing*. PAris.

Cura, Rémi, Julien Perret, and Nicolas Paparoditis (2015b). "Point Cloud Server (pcs): Point Clouds In-Base Management and Processing." In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* 1, pp. 531–539.

— (2016). "Implicit LOD for processing, visualisation and classification in Point Cloud Servers." In: *CoRR* abs/1602.06920.

Demantké, Jerome (2014). "Reconstruction of photorealistic 3D models of facades from terrestrial images and laser data." English. PhD thesis. Paris: Paris Est.

Despine, Guillaume and Caroline Baillard (2011). "Realistic Road Modelling for Driving Simulators using GIS Data." en. In: *Advances in Cartography and GIScience. Volume 2*. Ed. by Anne Ruas. Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg, pp. 431–448. ISBN: 978-3-642-19213-5 978-3-642-19214-2.

Deussen, Oliver, Pat Hanrahan, Bernd Lintermann, Radomír M\vech, Matt Pharr, and Przemyslaw Prusinkiewicz (1998). "Realistic modeling and rendering of plant ecosystems." In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, pp. 275–286.

Djaouti, Damien, Julian Alvarez, and Jean-Pierre Jessel (2011). "Classifying serious games: the G/P/S model." In: *Handbook of research on improving learning and motivation through educational games: Multidisciplinary approaches*, pp. 118–136. (Visited on 05/12/2016).

Duda, Richard O. and Peter E. Hart (1972). "Use of the Hough transformation to detect lines and curves in pictures." In: *Communications of the ACM* 15.1, pp. 11–15. ISSN: 00010782. DOI: 10.1145/361237.361242.

Duives, Dorine C., Winnie Daamen, and Serge P. Hoogendoorn (2013). "State-of-the-art crowd motion simulation models." In: *Transportation Research Part C: Emerging Technologies* 37, pp. 193–209. ISSN: 0968090X. DOI: 10.1016/j.trc.2013.02.005.

Edelsbrunner, H., D. Kirkpatrick, and R. Seidel (1983). "On the shape of a set of points in the plane." In: *IEEE Transactions on Information Theory* 29.4, pp. 551–559. ISSN: 0018-9448. DOI: 10.1109/TIT.1983.1056714.

Eitz, Mathias, Ronald Richter, Tamy Boubekeur, Kristian Hildebrand, and Marc Alexa (2012). "Sketch-Based Shape Retrieval." In: *ACM Trans. Graph. (Proc. SIGGRAPH)* 31.4.

Elseberg, Jan, Dorit Borrmann, and Andreas Nüchter (2013). "One billion points in the cloud – an octree for efficient processing of 3D laser scans." In: *ISPRS Journal of Photogrammetry and Remote Sensing*. Terrestrial 3D modelling 76, pp. 76–88. ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2012.10.004.

Ester, Martin, Hans-peter Kriegel, Jörg S, and Xiaowei Xu (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise." en. In: proceedings of 2nd International Conference on Knowledge Discovery and Data Mining. AAAI Press, pp. 226–231.

European Union Road Federation (2012). *European road statistic 2012*. English. Tech. rep. Belgium: ERF, p. 88.

Feng, Jiashi, Huan Xu, and Shuicheng Yan (2013). "Online robust pca via stochastic optimization." In: *Advances in Neural Information Processing Systems*, pp. 404–412.

Feng, Jun and Toyohide Watanabe (2014). *Index and Query Methods in Road Networks*. en. Springer. ISBN: 978-3-319-10789-9.

Fisher, Matthew, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan (2012). "Example-based synthesis of 3d object arrangements." In: *ACM Transactions on Graphics (TOG)* 31.6, p. 135.

Fornasier, M. and H. Rauhut (2010). "Compressive sensing." In: *Handbook of Mathematical Methods in Imaging* 1, pp. 187–229.

Frankhauser, Pierre (2008). "Fractal Geometry for Measuring and Modelling Urban Patterns." In: *The Dynamics of Complex Urban Systems*. Ed. by Sergio Albeverio, Denise Andrey, Paolo Giordano, and Alberto Vancheri. Heidelberg: Physica-Verlag HD, pp. 213–243. ISBN: 978-3-7908-1936-6.

Frey, Brendan J. and Delbert Dueck (2007). "Clustering by passing messages between data points." In: *science* 315.5814, pp. 972–976.

Gal, R., O. Sorkine, N. J. Mitra, and D. Cohen-Or (2009). "iWIRES: an analyze-and-edit approach to shape manipulation." In: *ACM Transactions on Graphics (TOG)*. Vol. 28, p. 33.

Galin, E., A. Peytavie, N. Maréchal, and E. Guérin (2010). "Procedural generation of roads." In: *Computer Graphics Forum*. Vol. 29, pp. 429–438.

Galin, E., A. Peytavie, E. Guérin, and B. Beneš (2011). "Authoring Hierarchical Road Networks." In: *Computer Graphics Forum*. Vol. 30, pp. 2021–2030.

Garcia-Dorado, Ignacio and Daniel G. Aliaga (2013). "Automatic modeling of planar-hinged buildings." In: *Eurographics 2013-Short Papers*, pp. 89–92.

Girardeau-Montaut, Daniel (2014). *CloudCompare*.

Golovinskiy, A., V. G Kim, and T. Funkhouser (2009). "Shape-based recognition of 3d point clouds in urban environments." In: *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 2154–2161.

Gong, Peng (2002). "Photo Ecometrics for Natural Resource Monitoring." en. In: *Deposit and Geoenvironmental Models for Resource Exploitation and Environmental Security*. Ed. by Andrea G. Fabbri, Gabor Gaál, and Richard B. McCammon. Nato Science Partnership Subseries: 2 (closed) 80. Springer Netherlands, pp. 65–80. ISBN: 978-1-4020-0990-7 978-94-010-0303-2.

Green, Peter J. (1995). "Reversible jump Markov chain Monte Carlo computation and Bayesian model determination." In: *Biometrika* 82.4, pp. 711–732.

Grzesiak-Kopec, K. and M. Ogorzalek (2013). "Intelligent 3D layout design with shape grammars." In: *2013 The 6th International Conference on Human System Interaction (HSI)*, pp. 265–270. DOI: 10.1109/HSI.2013.6577834.

Guennebaud, Gaël, Benoit Jacob, and others (2010). "Eigen v3." In:

Guillemot, Thierry, Andrès Almansa, and Tamy Boubekeur (2012). "Non Local Point Set Surfaces." In: *Proceedings of the International Conference on 3D Imaging, Moldeing, Processing, Visualization and Transmission (3DIMPVT)*.

Hagberg, Aric A., Daniel A. Schult, and Pieter J. Swart (2008). "Exploring network structure, dynamics, and function using NetworkX." In: *Proceedings of the 7th Python in Science Conference (SciPy2008)*. Pasadena, CA USA, pp. 11–15.

Hakala, Teemu, Juha Suomalainen, Sanna Kaasalainen, and Yuwei Chen (2012). "Full waveform hyperspectral LiDAR for terrestrial laser scanning." In: *Optics Express* 20.7, pp. 7119–7127.

Hatger, Carsten and Claus Brenner (2003). "Extraction of road geometry parameters from laser scanning and existing databases." In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 34.3/W13, pp. 225–230.

Hervieu, A., B. Soheilian, and M. Brédif (2015). "Road Marking Extraction Using a model&data driven Rj-Mcmc." In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2.3, p. 47.

Hervieu, Alexandre and Bahman Soheilian (2013). "Semi-automatic road/pavement modeling using mobile laser scanning." English. In: *ISPRS*. City Models, Roads and Traffic Volume II-3/W3, p. 31.

Hofle, Bernhard (2007). "Detection and utilization of the information potential of airborne laser scanning point cloud and intensity data by developing a management and analysis system." PhD thesis. Institute of Photogrammetry and Remote Sensing, Vienna University of Technology.

Hofmann, Sabine and Claus Brenner (2009). "Quality assessment of automatically generated feature maps for future driver assistance systems." In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '09. New York, NY, USA: ACM, pp. 500–503. ISBN: 978-1-60558-649-6. DOI: 10.1145/1653771.1653854.

Hornung, Armin, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard (2013). "OctoMap: an efficient probabilistic 3D mapping framework based on octrees." en. In: *Autonomous Robots* 34.3, pp. 189–206. ISSN: 0929-5593, 1573-7527. DOI: 10.1007/s10514-012-9321-0.

Huang, Yan, Jingliang Peng, C.-C. Jay Kuo, and M. Gopi (2006). "Octree-based progressive geometry coding of point clouds." In: *Proceedings of the 3rd Eurographics/IEEE VGTC conference on Point-Based Graphics*. Eurographics Association, pp. 103–110.

Hug, Ch, P. Krzystek, and W. Fuchs (2004). "Advanced lidar data processing with LasTools." In: *XXth ISPRS Congress*, pp. 12–23.

Hyvärinen, Aapo and Erkki Oja (2000). "Independent component analysis: algorithms and applications." In: *Neural networks* 13.4, pp. 411–430.

IGN (2014). *ITowns*. Paris.

IQmulus (2014). *IQmulus & TerraMobilita Contest*. Paris.

Ijiri, T., R. Mech, T. Igarashi, and G. Miller (2008). "An Example-based Procedural System for Element Arrangement." In: *Computer Graphics Forum*. Vol. 27, pp. 429–436.

Iovan, Corina, Paul-Henry Cournede, Thomas Guyard, Benoit Bayol, Didier Boldo, and Matthieu Cord (2013). "Model-Based Analysis–Synthesis for Realistic Tree Reconstruction and Growth Simulation." In: *IEEE Transactions on Geoscience and Remote Sensing*, pp. 1–1. ISSN: 0196-2892, 1558-0644. DOI: 10.1109/TGRS.2013.2251467.

Isenburg, Martin (2013). "LASzip." In: *Photogrammetric Engineering & Remote Sensing* 79.2, pp. 209–217.

Jain, Arjun, Thorsten Thormählen, Tobias Ritschel, and Hans-Peter Seidel (2012). "Material memex: automatic material suggestions for 3D objects." In: *ACM Trans. Graph.* 31.6, 143:1–143:8. ISSN: 0730-0301. DOI: 10.1145/2366145.2366162.

Jang, J., P. Wonka, W. Ribarsky, and C. D. Shaw (2006). "Punctuated simplification of man-made objects." In: *The Visual Computer* 22.2, pp. 136–145.

Jin, Hang, Yanming Feng, and Zhengrong Li (2009). "Extraction of Road Lanes from High-Resolution Stereo Aerial Imagery Based on Maximum Likelihood Segmentation and Texture Enhancement." In: *Digital Image Computing: Techniques and Applications, 2009.* Melbourne: IEEE, pp. 271–276. ISBN: 978-1-4244-5297-2. DOI: 10.1109/DICTA.2009.52.

Kalogerakis, E., D. Nowrouzezahrai, P. Simari, and K. Singh (2009). "Extracting lines of curvature from noisy point clouds." In: *Computer-Aided Design* 41.4, pp. 282–292.

Kelly, G. and H. McCabe (2006). "A survey of procedural techniques for city generation." In: *ITB Journal* 14, pp. 87–130.

Kiruthika, J. and S. Khaddaj (2014). "Performance Issues and Query Optimization in Big Multidimensional Data." In: *2014 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*, pp. 24–28. DOI: 10.1109/DCABES.2014.8.

Klavdianos, P., Q. Zhang, and E. Izquierdo (2013). "A concise survey for 3D reconstruction of building fa\ccades." In: *Image Analysis for Multimedia Interactive Services (WIAMIS), 2013 14th International Workshop on*, pp. 1–4.

Kolbe, Thomas H., Gerhard Gröger, and Lutz Plümer (2005). "CityGML – Interoperable Access to 3D City Models." In: *Proceedings of the first International Symposium on Geo-Information for Disaster Management*. Springer Verlag, pp. 21–23.

Krecklau, L. and L. Kobbelt (2011). "Procedural modeling of interconnected structures." In: *Computer Graphics Forum*. Vol. 30, pp. 335–344.

Krecklau, Lars, Christopher Manthei, and Leif Kobbelt (2012). "Procedural Interpolation of Historical City Maps." In: *EUROGRAPHICS* 31.2.

Krecklau, Lars, Darko Pavic, and Leif Kobbelt (2010). "Generalized Use of Non-Terminal Symbols for Procedural Modeling." In: *Comput. Graph. Forum* 29.8, pp. 2291–2303.

Kumar, Pankaj, Conor P. McElhinney, Paul Lewis, and Timothy McCarthy (2013). "An automated algorithm for extracting road edges from terrestrial mobile LiDAR data." In: *ISPRS Journal of Photogrammetry and Remote Sensing* 85, pp. 44–55. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2013.08.003.

Kuntzsch, Colin, Monika Sester, and Claus Brenner (2015). "Generative models for road network reconstruction." en. In: *International Journal of Geographical Information Science*, pp. 1–28. ISSN: 1365-8816, 1362-3087. DOI: 10.1080/13658816.2015.1092151.

Labatut, P., J. P. Pons, and R. Keriven (2009). "Hierarchical shape-based surface reconstruction for dense multi-view stereo." In: *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pp. 1598–1605.

Lafarge, F., R. Keriven, M. Brédif, and V. H. Hiep (2010). "Hybrid multi-view reconstruction by jump-diffusion." In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 350–357.

Lafarge, F., R. Keriven, M. Brédif, and Vu Hoang-Hiep (2013). "A Hybrid Multiview Stereo Algorithm for Modeling Urban Scenes." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.1, pp. 5–17. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2012.84.

Lasram, Anass, Sylvain Lefebvre, and Cyrille Damez (2012). "Scented Sliders for Procedural Textures." In: *Computer Graphics Forum (Eurographics conf. proc.)*

Lau, Manfred, Akira Ohgawara, Jun Mitani, and Takeo Igarashi (2011). "Converting 3D furniture models to fabricatable parts and connectors." In: ACM Press, p. 1. ISBN: 978-1-4503-0943-1. DOI: 10.1145/1964921.1964980.

Leclercq, Ludovic, Jorge Andres Laval, and Estelle Chevallier (2007). "The Lagrangian Coordinates and What it Means for First Order Traffic Flow Models." In: ISBN: 978-0-08-045375-0.

Lewis, P., C. P. Mc Elhinney, and T. McCarthy (2012). "LiDAR data management pipeline; from spatial database population to web-application visualization." In: *Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications*, p. 16.

Li, Yangyan, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra (2011). "GlobFit : Consistently Fitting Primitives by Discovering Global Relations." In: ACM Press, p. 1. ISBN: 978-1-4503-0943-1. DOI: 10.1145/1964921.1964947.

Li, Yangyan, Xiaochen Fan, Niloy J. Mitra, Daniel Chamovitz, Daniel Cohen-Or, and Baoquan Chen (2013). "Analyzing Growing Plants from 4D Point Cloud Data." In: *ACM Trans. Graph.* 32.6, 157:1–157:10. ISSN: 0730-0301. DOI: 10.1145/2508363.2508368.

Lin, Hui, Jizhou Gao, Yu Zhou, Guiliang Lu, Mao Ye, Chenxi Zhang, Ligang Liu, and Ruigang Yang (2013). "Semantic Decomposition and Reconstruction of Residential Scenes from LiDAR Data." In: *ACM Transactions on Graphics,(Proc. of SIGGRAPH 2013)* 32.4.

Lintermann, Bernd and Oliver Deussen (1999). "Interactive modeling of plants." In: *Computer Graphics and Applications, IEEE* 19.1, pp. 56–65.

Lipp, M., D. Scherzer, P. Wonka, and M. Wimmer (2011). "Interactive modeling of city layouts using layers of procedural content." In: *Computer Graphics Forum*. Vol. 30, pp. 345–354.

Lippow, M. A., L. P. Kaelbling, and T. Lozano-Perez (2008). "Learning grammatical models for object recognition." In: *Logic and Probability for Scene Interpretation, ser. Dagstuhl Seminar Proceedings*.

Livny, Yotam, Feilong Yan, Matt Olson, Baoquan Chen, Hao Zhang, and Jihad El-Sana (2010). "Automatic reconstruction of tree skeletal structures from point clouds." In: *ACM Transactions on Graphics (TOG)*. Vol. 29. ACM, p. 151.

Livny, Yotam, Soeren Pirk, Zhanglin Cheng, Feilong Yan, Oliver Deussen, Daniel Cohen-Or, and Baoquan Chen (2011). "Texture-lobes for Tree Modelling." In: *ACM SIGGRAPH 2011 Papers*. SIGGRAPH '11. New York, NY, USA: ACM, 53:1–53:10. ISBN: 978-1-4503-0943-1. DOI: 10.1145/1964921.1964948.

Lobo, María-Jesús, Emmanuel Pietriga, and Caroline Appert (2015). "An Evaluation of Interactive Map Comparison Techniques." en. In: *CHI '15 Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM Press, pp. 3573–3582. ISBN: 978-1-4503-3145-6. DOI: 10.1145/2702123.2702130.

Mackaness, William, Dirk Burghardt, and Cécile Duchêne (2014). "Map Generalisation: Fundamental to the Modelling and Understanding of Geographic Space." en. In: *Abstracting Geographic Information in a Data Rich World*. Ed. by Dirk Burghardt, Cécile Duchêne, and William Mackaness. Lecture Notes in Geoinformation and Cartography. Springer International Publishing, pp. 1–15. ISBN: 978-3-319-00202-6 978-3-319-00203-3. DOI: 10.1007/978-3-319-00203-3_1.

Mallet, Clément (2010). "Analyse de données lidar à Retour d'Onde Complète pour la classification en milieu urbain." anglais. PhD thesis. Telecom Paristech IGN Matis.

Martinez-Rubi, Oscar, Martin L. Kersten, Romulo Goncalves, and Milena Ivanova (2014). "A column-store meets the point clouds." In: *FOSS4G-Europe Academic Track*.

Martinez-Rubi, Oscar, Peter van Oosterom, Romulo Gonçalves, Theo Tijssen, Milena Ivanova, Martin L. Kersten, and Foteini Alvanaki (2015). "Benchmarking and improving point cloud data management in MonetDB." In: *SIGSPATIAL Special* 6.2, pp. 11–18.

Martinovic, Andelo and Luc Van Gool (2013). "Bayesian Grammar Learning for Inverse Procedural Modeling." In: *CVPR, 2013*. IEEE, pp. 201–208. ISBN: 978-0-7695-4989-7. DOI: 10.1109/CVPR.2013.33.

Mastoropoulou, Georgia, Kurt Debattista, Alan Chalmers, and Tom Troscianko (2005). "The influence of sound effects on the perceived smoothness of rendered animations." In: *Proceedings of the 2nd symposium on Applied perception in graphics and visualization*. ACM, pp. 9–15.

McCrae, J. and K. Singh (2009a). "Sketch-based path design." In: *Proceedings of the Graphics Interface 2009 Conference*. Kelowna, British Columbia, Canada: Canadian Information Processing Society, pp. 95 –102.

— (2009b). "Sketching piecewise clothoid curves." In: *Computers & Graphics* 33.4, pp. 452–461.

McKay, M. D., R. J. Beckman, and W. J. Conover (1979). "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code." In: *Technometrics* 21.2, p. 239. ISSN: 00401706. DOI: 10.2307/1268522.

Meagher, Donald (1982). "Geometric modeling using octree encoding." In: *Computer graphics and image processing* 19.2, pp. 129–147.

Meng, Liqiu and Andrea Forberg (2007). "3D building generalisation." In: *Challenges in the Portrayal of Geographic Information. Elsevier Science, Amsterdam*, pp. 211–232.

Merrell, P. and D. Manocha (2011). "Model synthesis: A general procedural modeling algorithm." In: *Visualization and Computer Graphics, IEEE Transactions on* 99, pp. 1–1.

Mitra, N. J, L. J Guibas, and M. Pauly (2006). "Partial and approximate symmetry detection for 3D geometry." In: *ACM Transactions on Graphics (TOG)* 25.3, pp. 560–568.

Mitra, Niloy J., Mark Pauly, Michael Wand, and Duygu Ceylan (2012). "Symmetry in 3D Geometry: Extraction and Applications." In: *EUROGRAPHICS State-of-the-art Report*.

Mongus, domen, bojan rupnik, and Borut Zalik (2011). "Comparison of Algorithms for lossless LiDAR Data Compression." English. In: Geospatial Crossroads GI_Forum '11. [S.l.]: Wichmann, H. ISBN: ISBN 978-3-87907-509-6.

Montoya-Zegarra, Javier A., Jan D. Wegner, L 'ubor Ladicky, and Konrad Schindler (2014). "Mind the gap: modeling local and global context in (road) networks." In: *German Conference on Pattern Recognition (GCPR)*. Springer, pp. 212–223.

Moussafir, Jacques, Christophe Olry, Maxime Nibart, Armand Albergel, Patrick Armand, Christophe Duchenne, Frédéric Mahe, Ludovic Thobois, and O. Oldrini (2013). "Aircity, a very High-resolution 3D Atmospheric Dispersion Modeling System for Paris." In: *15th Int. Conf. on Harmonisation within Atmospheric Dispersion Modelling for Regulatory Purposes*.

Mueller, Andre, Michael Himmelsbach, Thorsten Luettel, Felix V. Hundelshausen, and Hans-Joachim Wuensche (2011). "GIS-based topological robot localization through LIDAR crossroad detection." In: *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*. IEEE, pp. 2001–2008.

Müller, P., T. Vereenooghe, P. Wonka, I. Paap, and L. Van Gool (2006). "Procedural 3D reconstruction of Puuc buildings in Xkipche." In: *Eurographics Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)*, pp. 139–146.

Musialski, Przemyslaw and Michael Wimmer (2013). "Inverse-Procedural Methods for Urban Models." In: *Proc. of 1st Eurographics Workshop on Urban Data Modelling and Visualisation*. Ed. by V. Tourre and G. Besuievsky. Girona, Spain: Eurographics Association, pp. 31–32.

Musialski, Przemyslaw, Michael Wimmer, and Peter Wonka (2012). "Interactive Coherence-Based Façade Modeling." In: *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2012)* 31.2, to appear.

Musialski, Przemyslaw, Peter Wonka, Daniel G. Aliaga, Michael Wimmer, Luc van Gool, and Werner Purgathofer (2012). "A Survey of Urban Reconstruction." In: *EUROGRAPHICS 2012 State of the Art Reports*. EG STARs. Eurographics Association, pp. 1–28.

Neteler, M., M.H. Bowman, M. Landa, and M. Metz (2012). "GRASS GIS: a multi-purpose Open Source GIS." In: *Environmental Modelling & Software* 31, pp. 124–130. DOI: 10.1016/j.envsoft.2011.11.014.

Nguyen, H.h., B. Desbenoit, and M. Daniel (2014). "Realistic road path reconstruction from GIS data." en. In: *Computer Graphics Forum* 33.7, pp. 259–268. ISSN: 1467-8659. DOI: 10.1111/cgf.12494.

Niggeler, Laurent (2009). *Genève en 3D: une nouvelle dimension pour gérer son territoire (Genf erschliesst mit 3D eine neue Dimension für die Verwaltung seines Hoheitsgebiets),(3D Geneva : a new dimension to manage territory)*. fr,de. http://www.cadastre.ch/internet/cadastre/fr/home/docu/publication/F006.html.

OECD (2010). *Tackling Inequalities in Brazil, China, India and South Africa*. en. Paris: Organisation for Economic Co-operation and Development. ISBN: 978-92-64-08835-1.

Oscar Martinez-Rubi, Stefan Verhoeven, Maarten Van Meersbergen, Markus Schütz, Peter Van Oosterom, Romulo Goncalves, and Theo Tijssen (2015). "Taming the beast: Free and open-source massive point cloud web visualization." In: DOI: 10.13140/RG.2.1.1731.4326.

Otepka, J., G. Mandlburger, and W. Karel (2012). "The OPALS Data Manager—Efficient Data Management for Processing Large Airborne Laser Scanning Projects." In: *Proceedings of the ISPRS Annals of the Photogrammetry, Melbourne, Australia* 25, pp. 153–159.

Otepka, Johannes, Sajid Ghuffar, Christoph Waldhauser, Ronald Hochreiter, and Norbert Pfeifer (2013). "Georeferenced Point Clouds: A Survey of Features and Point Cloud Management." en. In: *ISPRS International Journal of Geo-Information* 2.4, pp. 1038–1065. ISSN: 2220-9964. DOI: 10.3390/ijgi2041038.

Papadakis, P., I. Pratikakis, S. Perantonis, and T. Theoharis (2007). "Efficient 3D shape matching and retrieval using a concrete radialized spherical projection representation." In: *Pattern Recognition* 40.9, pp. 2437–2452.

Paparoditis, Nicolas, Daniela Craciun, and Francis Schmitt (2012). "Image-Laser Fusion for In Situ 3D Modeling of Complex Environments: A 4D Panoramic-Driven Approach." In:

Paparoditis, Nicolas, Jean-Pierre Papelard, Bertrand Cannelle, Alexandre Devaux, Bahman Soheilian, Nicolas David, and Erwann Houzay (2012). "Stereopolis II: A multi-purpose and multi-sensor 3D mobile mapping system for street visualisation and 3D metrology." In: *Revue française de photogrammétrie et de télédétection* 200.1, pp. 69–79.

Parish, Y. I. H. and P. Müller (2001). "Procedural modeling of cities." In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 301–308.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Pindat, Cyprien, Emmanuel Pietriga, Olivier Chapuis, and Claude Puech (2012). "JellyLens: content-aware adaptive lenses." In: *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, pp. 261–270.

Pirk, S., O. Stava, J. Kratt, M. A. M. Said, B. Neubert, R. M\vech, B. Benes, and O. Deussen (2012). "Plastic trees: interactive self-adapting botanical tree models." In: *ACM Transactions on Graphics (TOG)* 31.4, p. 50.

Poullis, Charalambos and Suya You (2010). "Delineation and geometric modeling of road networks." In: *ISPRS Journal of Photogrammetry and Remote Sensing* 65.2, pp. 165–181. ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2009.10.004.

Preiner, Reinhold, Oliver Mattausch, Murat Arikan, Renato Pajarola, and Michael Wimmer (2014). "Continuous Projection for Fast L-1 Reconstruction." In: *ACM Transactions on Graphics (TOG)* 33.4, p. 47.

Preuksakarn, C., F. Boudon, P. Ferraro, J. B Durand, E. Nikinmaa, and C. Godin (2010). "Reconstructing plant architecture from 3D laser scanner data." In:

Quackenbush, Lindi J, Iungho Im, and Yue Zuo (2013). "Road extraction: a review of LiDAR-focused studies." In: *Remote Sensing of Natural Resources*, pp. 155–169.

Rainville, De, Christian Gagné, Olivier Teytaud, Denis Laurendeau, and others (2012). "Evolutionary optimization of low-discrepancy sequences." In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 22.2, p. 9.

Ramilo, Runddy (2005). *The Visual Perception and Human Cognition of Urban Environments Using Semantic Scales*. Tech. rep.

Ravanbakhsh, M. and C. S. Fraser (2009). "Road roundabout extraction from very high resolution aerial imagery." In: *International Archives of Photogrametry and Remote Sensing*.

Richter, Rico and Jürgen Döllner (2014). "Concepts and techniques for integration, analysis and visualization of massive 3D point clouds." en. In: *Computers, Environment and Urban Systems* 45, pp. 114–124. ISSN: 01989715. DOI: 10.1016/j.compenvurbsys.2013.07.004.

Rieg, Lorenzo, Volker Wichmann, Martin Rutzinger, Rudolf Sailer, Thomas Geist, and Johann Stötter (2014). "Data infrastructure for multitemporal airborne LiDAR point cloud analysis – Examples from physical geography in high mountain environments." In: *Computers, Environment and Urban Systems* 45, pp. 137–146. ISSN: 0198-9715. DOI: 10.1016/j.compenvurbsys.2013.11.004.

Runions, A., B. Lane, and P. Prusinkiewicz (2007). "Modeling trees with a space colonization algorithm." In: *Eurographics Association*, pp. 63–70.

Rusinkiewicz, Szymon and Marc Levoy (2000). "QSplat: A multiresolution point rendering system for large meshes." In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., pp. 343–352.

Rusu, Radu Bogdan and Steve Cousins (2011). "3d is here: Point cloud library (pcl)." In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, pp. 1–4.

SETRA (2006). *Comprendre les principaux paramètres de conception géométrique des routes*. Français. Tech. rep., p. 30.

Sabo, Nouri, A Beaulieu, D Bélanger, Y Belzile, and B Piché (2014). *The GeoHashTree: a multi-resolution data structure for the management of point clouds*. English. Technical notes 4. canada.

Sahr, Kevin (2011). "Hexagonal discrete global grid systems for geospatial computing." In: *Archives of Photogrammetry, Cartography and Remote Sensing* 22, pp. 363–376.

Schnabel, Ruwen and Reinhard Klein (2006). "Octree-based Point-Cloud Compression." In: *SPBG*, pp. 111–120.

Schütz, Markus and Michael Wimmer (2015). *High-Quality Point Based Rendering Using Fast Single Pass Interpolation*. IEEE. ISBN: 978-1-5090-0048-7.

Schwartges, Nadine, Dennis Allerkamp, Jan-Henrik Haunert, and Alexander Wolff (2013). "Optimizing Active Ranges for Point Selection in Dynamic Maps." In: *Proceedings of the 16th ICA Generalisation Workshop (ICA'13)*. Dresden.

Serna, Andrés and Beatriz Marcotegui (2013). "Urban accessibility diagnosis from mobile laser scanning data." In: *ISPRS Journal of Photogrammetry and Remote Sensing* 84, pp. 23–32. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2013.07.001.

Serna, Andres and Beatriz Marcotegui (2014 (in print)). "Detection, segmentation and classification of 3D urban objects using mathematical morphology and supervised learning." English. In: *ISPRS Journal of Photogrammetry and Remote Sensing*, p. 34.

Sester, Monika (2001). *Kohonen Feature Nets for Typification*. English. Beijing.

Seto, Karen C., Michail Fragkias, Burak Güneralp, and Michael K. Reilly (2011). "A Meta-Analysis of Global Urban Land Expansion." In: *PLoS ONE* 6.8, e23777. DOI: 10.1371/journal.pone.0023777.

Shao, T., W. Xu, K. Yin, J. Wang, K. Zhou, and B. Guo (2011). "Discriminative Sketch-based 3D Model Retrieval via Robust Shape Matching." In: *Computer Graphics Forum*. Vol. 30.

Shao, Tianjia, Weiwei Xu, Kun Zhou, Jingdong Wang, Dongping Li, and Baining Guo (2012). "An interactive approach to semantic modeling of indoor scenes with an RGBD camera." In: *ACM Trans. Graph.* 31.6, 136:1–136:11. ISSN: 0730-0301. DOI: 10.1145/2366145.2366155.

Shapira, L., S. Shalom, A. Shamir, D. Cohen-Or, and H. Zhang (2009). "Contextual Part Analogies in 3D Objects." In: *International Journal of Computer Vision* 89.2-3, pp. 309–326. ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-009-0279-0.

Shapovalov, Roman, Alexander Velizhev, and Olga Barinova (2010). "Non-associative markov networks for 3D point cloud classification." In: *Photogrammetric Computer Vision and Image Analysis (PCV 2010)*. Vol. 38, pp. 103–108.

Shen, Chao-Hui, Hongbo Fu, Kang Chen, and Shi-Min Hu (2012). "Structure recovery by part assembly." In: *ACM Trans. Graph.* 31.6, 180:1–180:11. ISSN: 0730-0301. DOI: 10.1145/2366145.2366199.

Shlens, Jonathon (2014). "A tutorial on principal component analysis." In: *arXiv preprint arXiv:1404.1100*.

Soheilian, B., O. Tournaire, N. Paparoditis, B. Vallet, and J.-P. Papelard (2013). "Generation of an integrated 3D city model with visual landmarks for autonomous navigation in dense urban areas." In: *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 304–309. DOI: 10.1109/IVS.2013.6629486.

Soheilian, Bahman and Lionel Atty (2016). *Rapport sur la modelisation fine de route*. Français.

Soheilian, Bahman, Nicolas Paparoditis, and Didier Boldo (2010). "3D road marking reconstruction from street-level calibrated stereo pairs." en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 65.4, pp. 347–359. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2010.03.003.

Soheilian, Bahman, Nicolas Paparoditis, and Bruno Vallet (2013). "Detection and 3D reconstruction of traffic signs from multiple view color images." en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 77, pp. 1–20. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2012.11.009.

Steiniger, Stefan and Robert Weibel (2007). "Relations among map objects in cartographic generalization." In: *Cartography and Geographic Information Science* 34.3, pp. 175–197.

Stuckler, J., Nenad Biresev, and Sven Behnke (2012). "Semantic mapping using object-class segmentation of RGB-D images." In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, pp. 3005–3010.

Talton, Jerry O., Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun (2011). "Metropolis procedural modeling." In: *ACM Trans. Graph.* 30.2, 11:1–11:14. ISSN: 0730-0301. DOI: 10.1145/1944846.1944851.

The CGAL Project (2015). *CGAL User and Reference Manual*. 4.6. CGAL Editorial Board.

Touya, Guillaume (2010). "A road network selection process based on data enrichment and structure detection." In: *Transactions in GIS* 14.5, pp. 595–614.

Touya, Guillaume, Bénédicte Bucher, Gilles Falquet, Kusay Jaara, and Stephan Steiniger (2014). "Modelling Geographic Relationships in Automated Environments." English. In: *Abstracting Geographic Information in a Data Rich World, Methodologies and Applications of Map Generalisation*. Springer. Lecture Notes in Geoinformation and Cartography. Dirk Burghardt, · Cécile Duchêne, · William Mackaness.

Umetani, N., T. Igarashi, and N. J. Mitra (2012). "Guided exploration of physically valid shapes for furniture design." In: *ACM Transactions on Graphics* 31.4.

Ummenhofer, Benjamin and Thomas Brox (2015). "Global, Dense Multiscale Reconstruction for a Billion Points." In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1341–1349.

United Nations (2012). *World Urbanization Prospects: The 2011 Revision*. Tech. rep. United Nations.

Vanegas, Maria Carolina, Isabelle Bloch, and Jordi Inglada (2013). "Alignment and Parallelism for the Description of High-Resolution Remote Sensing Images." In: *IEEE Transactions on Geoscience and Remote Sensing* 51.6, pp. 3542–3557. ISSN: 0196-2892, 1558-0644. DOI: 10.1109/TGRS.2012.2225628.

Wallace, Andrew, Caroline Nichol, and Iain Woodhouse (2012). "Recovery of Forest Canopy Parameters by Inversion of Multispectral LiDAR Data." In: *Remote Sensing* 4.12, pp. 509–531. ISSN: 2072-4292. DOI: 10.3390/rs4020509.

Wang, Fusheng, Ablimit Aji, and Hoang Vo (2014). "High Performance Spatial Queries for Spatial Big Data: from Medical Imaging to GIS." In: *The SIGSPATIAL Special* 6.3, pp. 11–18.

Wang, Guangxing and Qihao Weng (2013). *Remote Sensing of Natural Resources*. en. CRC Press. ISBN: 978-1-4665-5692-8.

Wang, Jie, Gary Lawson, and Yuzhong Shen (2014). "Automatic high-fidelity 3D road network modeling based on 2D GIS data." In: *Advances in Engineering Software* 76, pp. 86–98. ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2014.06.005.

Watt, Michael S, Thomas Adams, Hamish Marshall, David Pont, John Lee, David Crawley, and Pete Watt (2013). "Modelling variation in Pinus radiata stem volume and outerwood stress-wave velocity from LiDAR metrics." In: *New Zealand Journal of Forestry Science* 43.1, p. 1. ISSN: 1179-5395. DOI: 10.1186/1179-5395-43-1.

Wedel, A., H. Badino, C. Rabe, H. Loose, U. Franke, and D. Cremers (2009). "B-Spline Modeling of Road Surfaces With an Application to Free-Space Estimation." In: *IEEE Transactions on Intelligent Transportation Systems* 10.4, pp. 572–583. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2009.2027223.

Wei, Li-Yi, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk (2009). "State of the Art in Example-based Texture Synthesis." In: *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association.

Weinmann, M., S. Urban, S. Hinz, B. Jutzi, and C. Mallet (2015). "Distinctive 2D and 3D features for automated large-scale scene analysis in urban areas." en. In: *Computers & Graphics* 49, pp. 47–57. ISSN: 00978493. DOI: 10.1016/j.cag.2015.01.006.

Wilkie, David, Jason Sewall, Ming C. Lin, and Ming C. Lin (2012). "Transforming GIS Data into Functional Road Models for Large-Scale Traffic Simulation." In: *IEEE Transactions on Visualization and Computer Graphics* 18.6, pp. 890–901. ISSN: 1077-2626. DOI: 10.1109/TVCG.2011.116.

Wolfermann, Axel, Wael KM Alhajyaseen, and Hideki Nakamura (2011). "Modeling speed profiles of turning vehicles at signalized intersections." In: *3rd International*

Conference on Road Safety and Simulation RSS2011, Transportation Research Board TRB, Indianapolis.

Wu, Jianguo (2011). "Improving the writing of research papers: IMRAD and beyond." In: *Landscape Ecology* 26.10, pp. 1345 –1349. ISSN: 0921-2973. DOI: 10.1007/s10980-011-9674-3.

Xfrog (2014). *Xfrog - 3D Trees and 3D Plants for CG Artists*. http://xfrog.com/.

Xu, K., H. Zhang, D. Cohen-Or, and B. Chen (2012). "Fit and diverse: Set evolution for inspiring 3D shape galleries." In: *ACM Transactions on Graphics (TOG)* 31.4, p. 57.

Yang, Bisheng, Lina Fang, and Jonathan Li (2013). "Semi-automated extraction and delineation of 3D roads of street scene from mobile laser scanning point clouds." en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 79, pp. 80–93. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2013.01.016.

Yang, Yong-Liang, Jun Wang, Etienne Vouga, and Peter Wonka (2013). "Urban Pattern: Layout Design by Hierarchical Domain Splitting." In: *ACM Trans. Graph.* 32.6, 181:1–181:12. ISSN: 0730-0301. DOI: 10.1145/2508363.2508405.

Yeh, Anthony G. O., Teng Zhong, and Yang Yue (2015). "Hierarchical polygonization for generating and updating lane-based road network information for navigation from road markings." In: *International Journal of Geographical Information Science* 0.0, pp. 1–25. ISSN: 1365-8816. DOI: 10.1080/13658816.2015.1014373.

Yeh, Yi-Ting, Lingfeng Yang, Matthew Watson, Noah D. Goodman, and Pat Hanrahan (2012). "Synthesizing open worlds with constraints using locally annealed reversible jump MCMC." In: *ACM Trans. Graph.* 31.4, 56:1–56:11. ISSN: 0730-0301. DOI: 10.1145/2185520.2185552.

Yirci, Murat, Mathieu Brédif, Julien Perret, and Nicolas Paparoditis (2013). "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*. ACM, p. 31.

Youn, Choonhan, Viswanath Nandigam, Minh Phan, David Tarboton, Nancy Wilkins-Diehr, Chaitan Baru, Christopher Crosby, Anand Padmanabhan, and Shaowen Wang (2014). "Leveraging XSEDE HPC Resources to Address Computational Challenges with High-resolution Topography Data." In: *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. XSEDE '14. New York, NY, USA: ACM, 59:1–59:2. ISBN: 978-1-4503-2893-7. DOI: 10.1145/2616498.2616564.

Yu, Lap-Fai, Sai-Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F. Chan, and Stanley J. Osher (2011a). "Make it home : Automatic Optimization of Furniture Arrangement." In: ACM Press, p. 1. ISBN: 978-1-4503-0943-1. DOI: 10.1145/1964921.1964981.

Yu, Z., C. Xu, J. Liu, O. C Au, and X. Tang (2011b). "Automatic object segmentation from large scale 3D urban point clouds through manifold embedded mode seeking." In: *Proceedings of the 19th ACM international conference on Multimedia*, pp. 1297–1300.

Zhang, Chenxi, Liang Wang, and Ruigang Yang (2010). "Semantic segmentation of urban scenes using dense depth maps." In: *Computer Vision–ECCV 2010*. Springer, pp. 708–721.

Zhang, Geng, Nanning Zheng, Chao Cui, Yuzhen Yan, and Zejian Yuan (2009). "An efficient road detection method in noisy urban environment." In: *Intelligent Vehicles Symposium, 2009 IEEE*. IEEE, pp. 556–561.

Zhang, Lijuan, Frank Thiemann, and Monika Sester (2010). "Integration of GPS traces with road map." In: *Proceedings of the second international workshop on computational transportation science*. ACM, pp. 17–22.

Zhang, Wende (2010). "LIDAR-based road and road-edge detection." In: *Intelligent Vehicles Symposium (IV), 2010 IEEE*. IEEE, pp. 845–848.

Zinoune, Clément, Philippe Bonnifait, and Javier Ibanez-Guzman (2012). "Detection of missing roundabouts in maps for Driving Assistance Systems." In: *Intelligent Vehicles Symposium (IV), 2012 IEEE*. IEEE, pp. 123–128.

pgPointCloud, Ramsey (2014). *pgPointCloud*.

team Networkx, dev. (2014). *Networkx*.

team PGRouting, dev. (2015). *PgRouting*.

team PostGIS Topology, dev. (2014). *PostGIS Topology*.

team PostGIS, dev. (2014). *PostGIS*.

team PostgreSQL, dev. (2014). *PostgreSQL*.

team SciPy, dev. (2014). *SciPy: Open source scientific tools for Python*.

team Scikit, dev. (2014). "scikit-image: image processing in Python." In: *PeerJ* 2, e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453.

van Oosterom, Peter, Oscar Martinez-Rubi, Milena Ivanova, Mike Horhammer, Daniel Geringer, Siva Ravada, Theo Tijssen, Martin Kodde, and Romulo Gonçalves (2015). "Massive point cloud data management: Design, implementation and execution of a point cloud benchmark." en. In: *Computers & Graphics* 49.Special Section on Processing Large Geospatial Data, pp. 92–125. ISSN: 00978493. DOI: 10.1016/j.cag.2015.01.007.

DECLARATION

The authors would like to thank reviewers for their suggestions, and colleagues for ideas and help, in particular G. Le Meur. This work was supported in part by an ANRT grant (20130042). Put your declaration here.

Paris, Avril 2016

Rémi Cura

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Final Version as of May 22, 2016 (`classicthesis` version 1.00).

GRAPHICAL TABLE OF FIGURE

CHAPTER 1

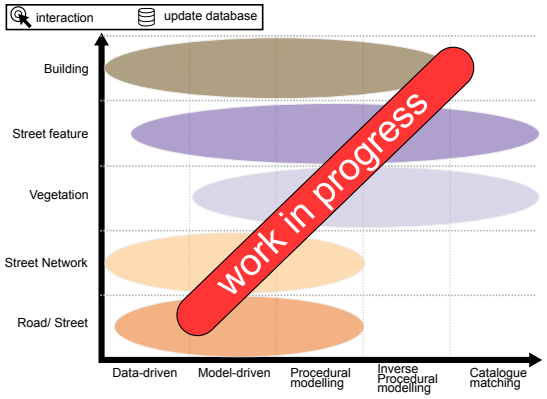


Figure 5 on page 9

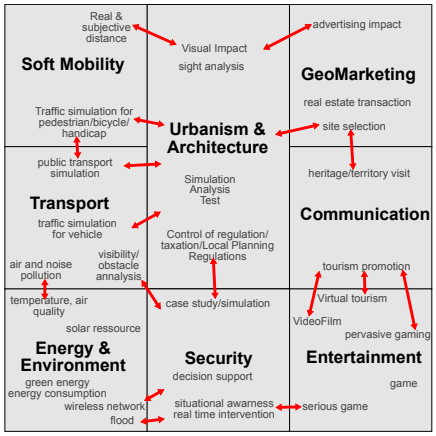


Figure 6 on page 12

Figure 7 on page 13

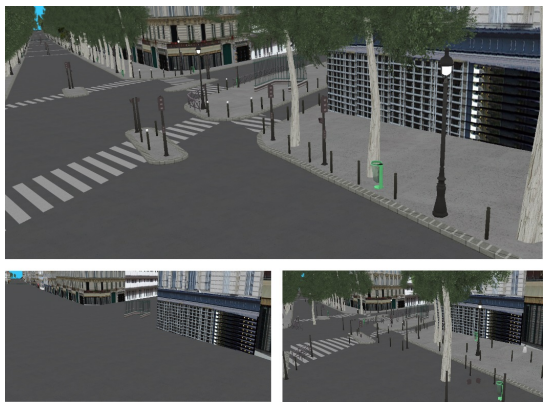


Figure 8 on page 15

Figure 9 on page 16

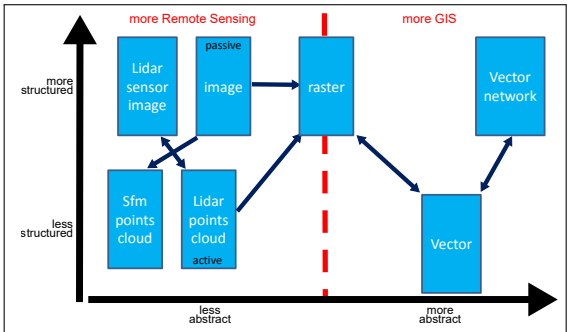


Figure 10 on page 17

Figure 11 on page 18

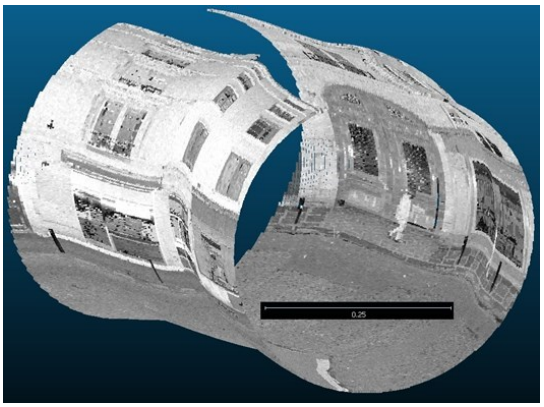


Figure 12 on page 19

Figure 13 on page 20

Figure 14 on page 21

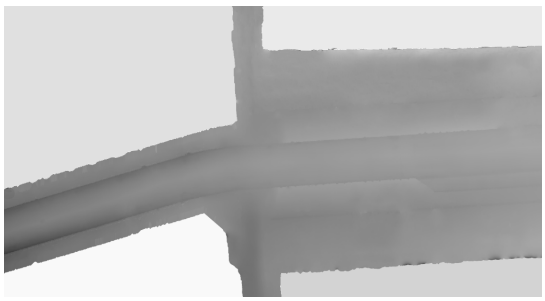


Figure 15 on page 22

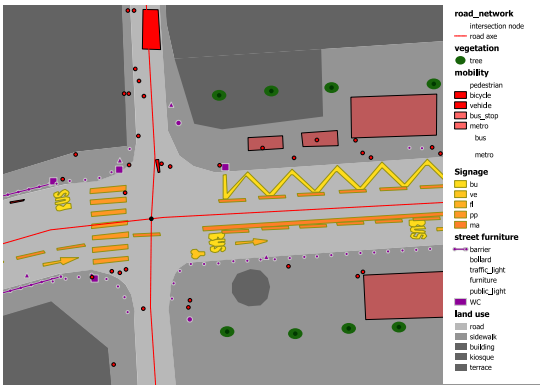


Figure 16 on page 23

Figure 17 on page 26

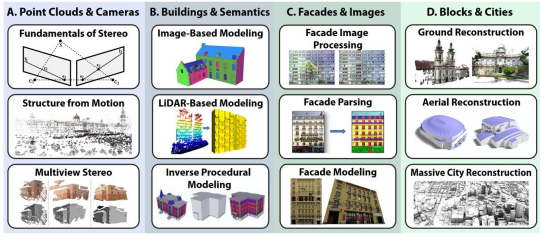


Figure 18 on page 27

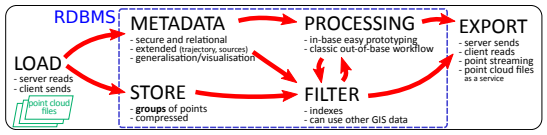


Figure 19 on page 52

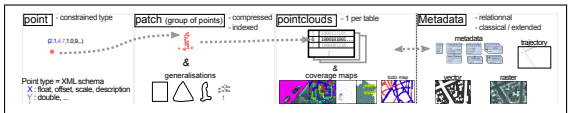


Figure 20 on page 57

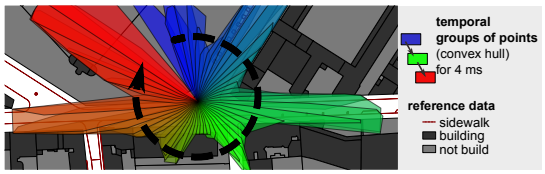


Figure 21 on page 58

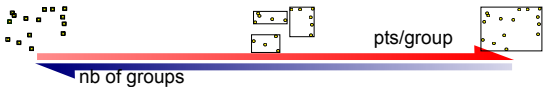


Figure 22 on page 59

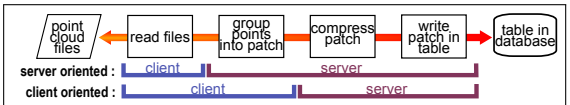


Figure 23 on page 59

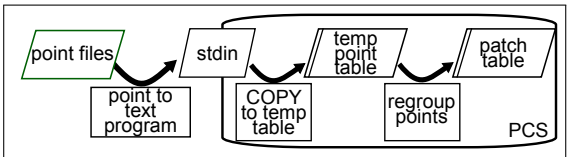


Figure 24 on page 60

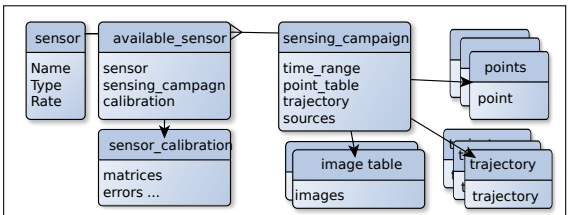


Figure 25 on page 61

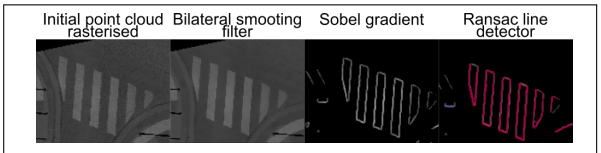


Figure 26 on page 62

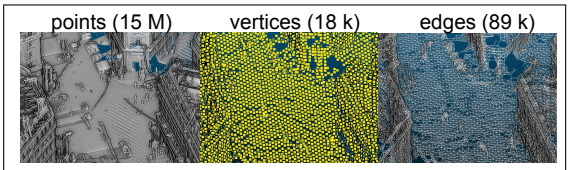


Figure 27 on page 63

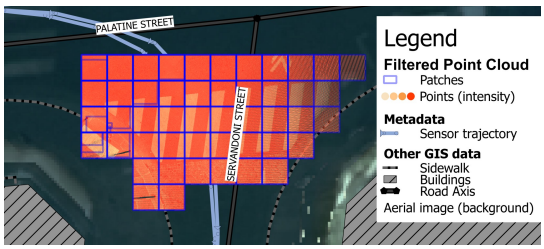


Figure 28 on page 63

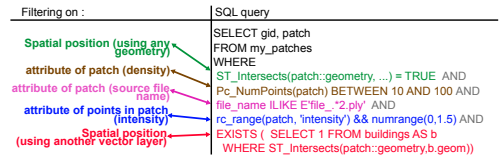


Figure 29 on page 64

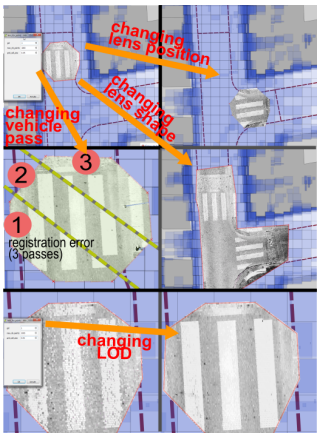


Figure 30 on page 66

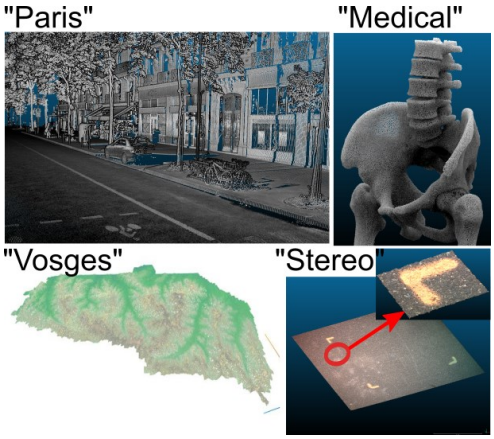


Figure 31 on page 68

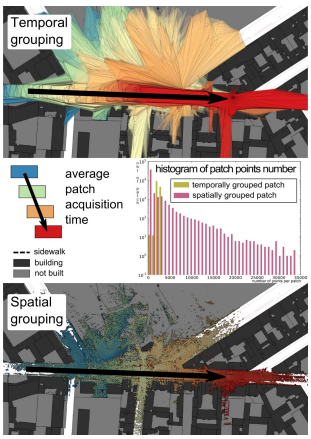


Figure 32 on page 71

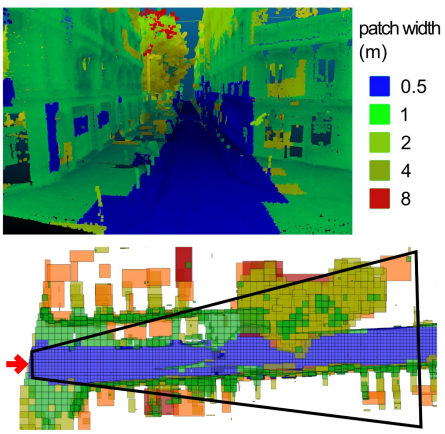


Figure 33 on page 72

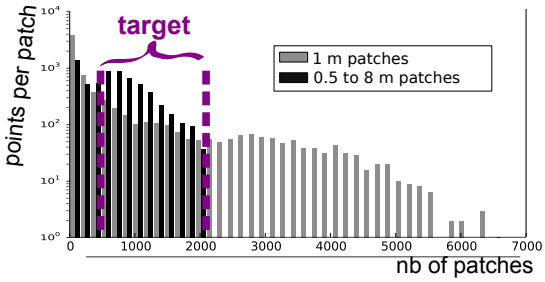


Figure 34 on page 72

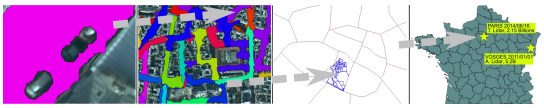


Figure 35 on page 74

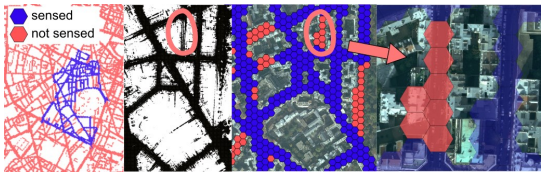


Figure 36 on page 75

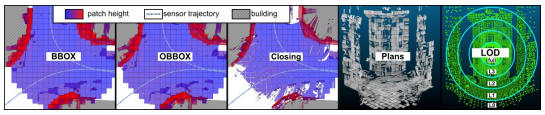


Figure 37 on page 76

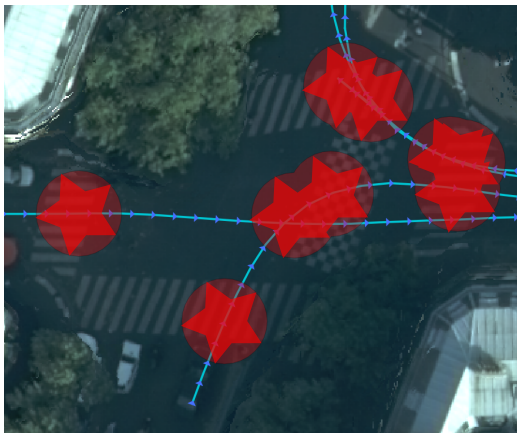


Figure 38 on page 77

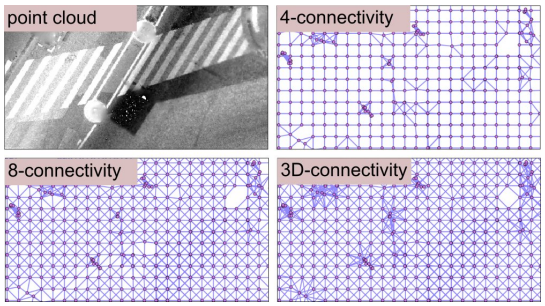


Figure 39 on page 78

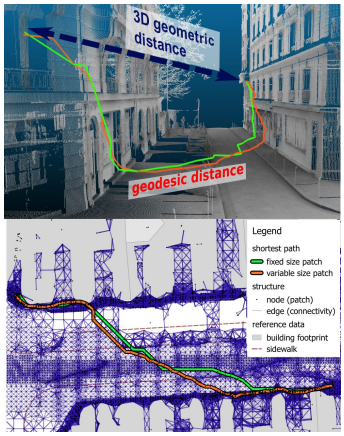


Figure 40 on page 78

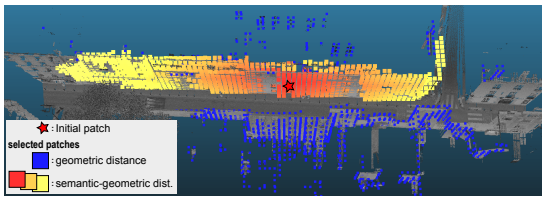


Figure 41 on page 79

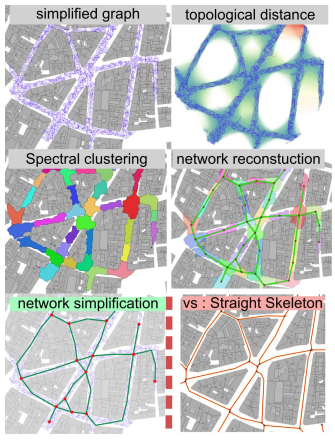


Figure 42 on page 80

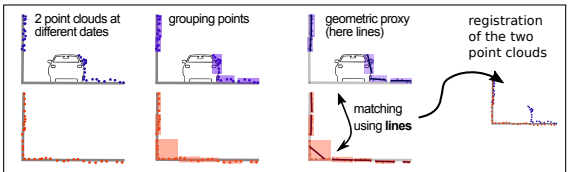


Figure 43 on page 86

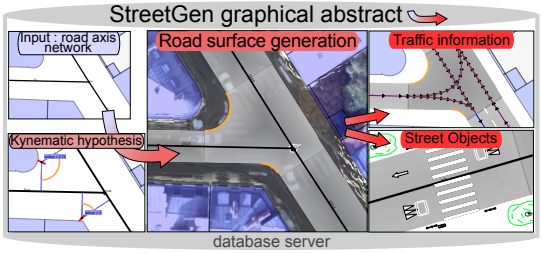


Figure 44 on page 89

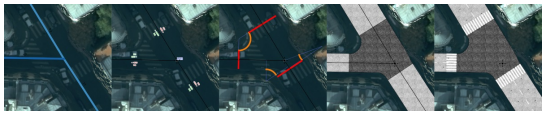


Figure 45 on page 90

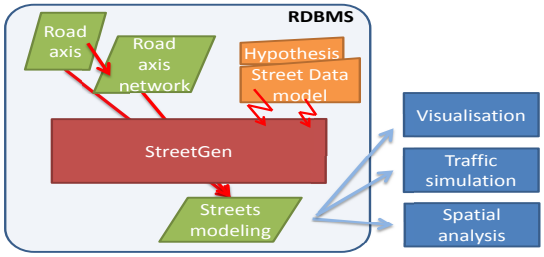


Figure 46 on page 92

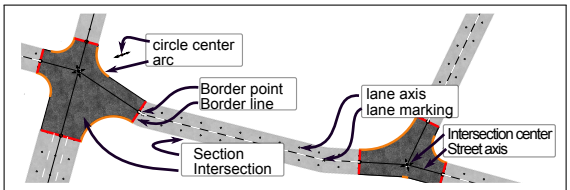


Figure 47 on page 93

Figure 48 on page 94

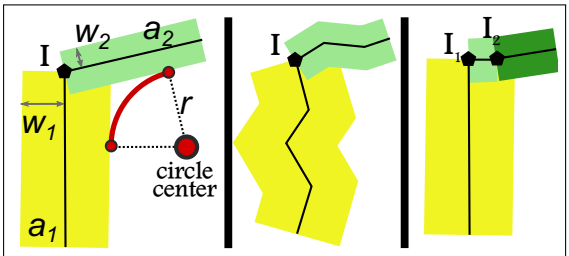


Figure 49 on page 95

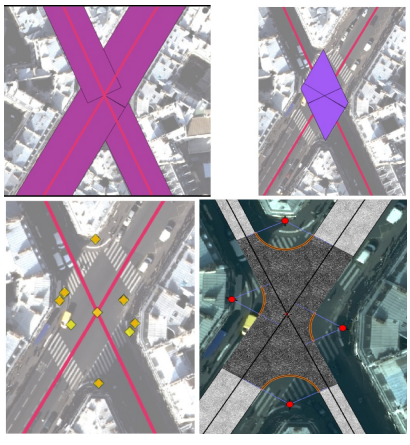


Figure 50 on page 96

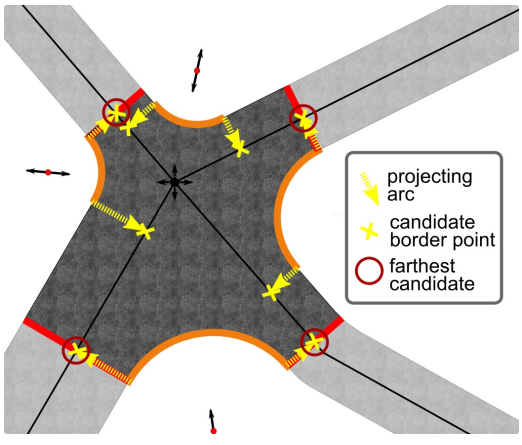


Figure 51 on page 97

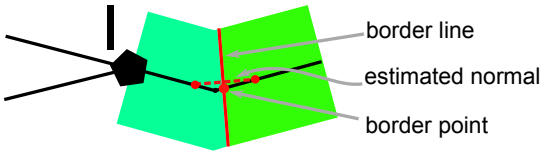


Figure 52 on page 98

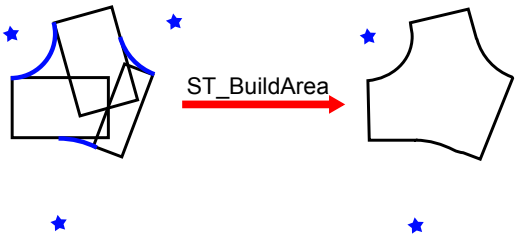


Figure 53 on page 98

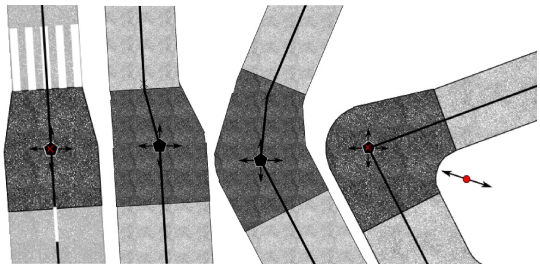


Figure 54 on page 98

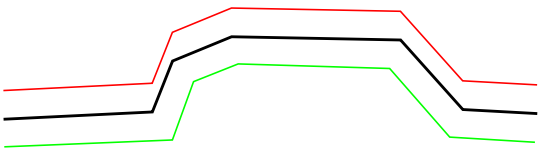


Figure 55 on page 99

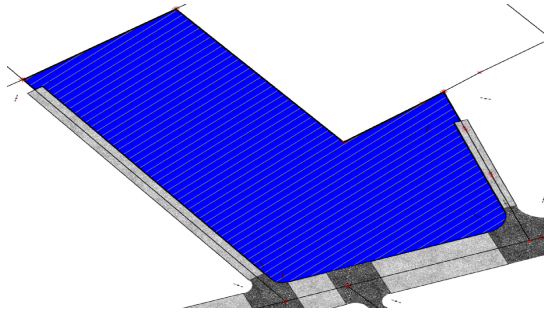


Figure 56 on page 100

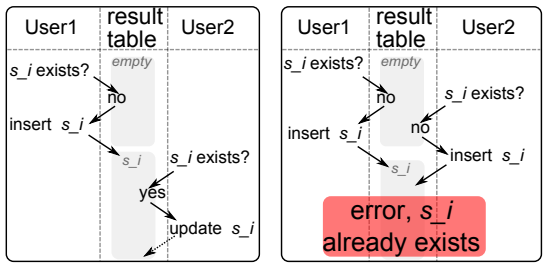


Figure 57 on page 101

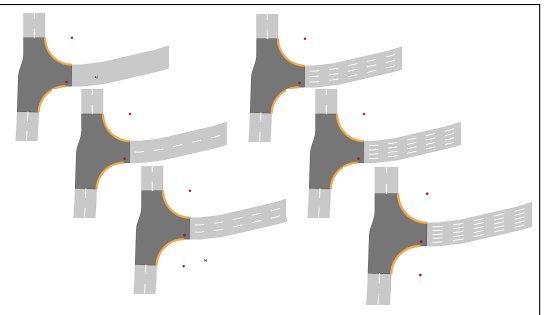


Figure 58 on page 103

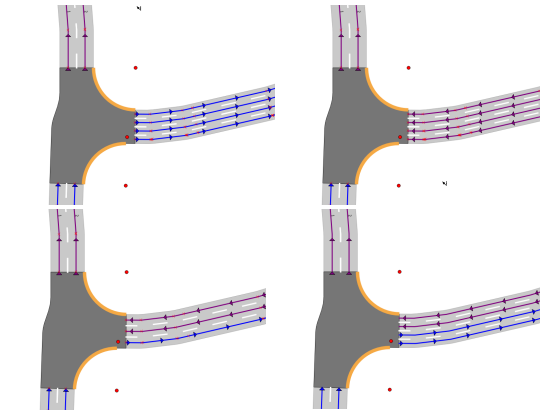


Figure 59 on page 103

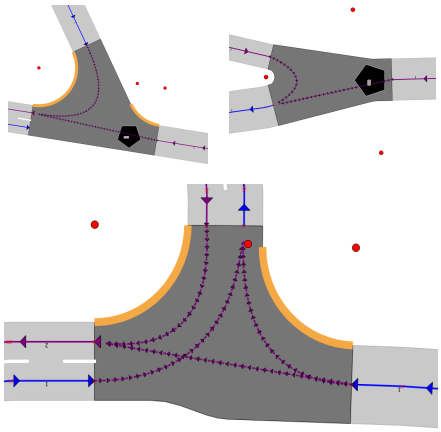


Figure 60 on page 104

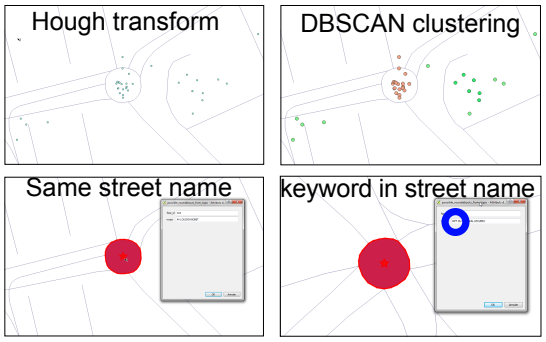


Figure 61 on page 106

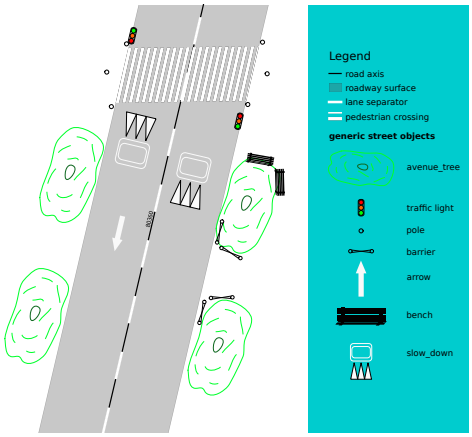


Figure 62 on page 107

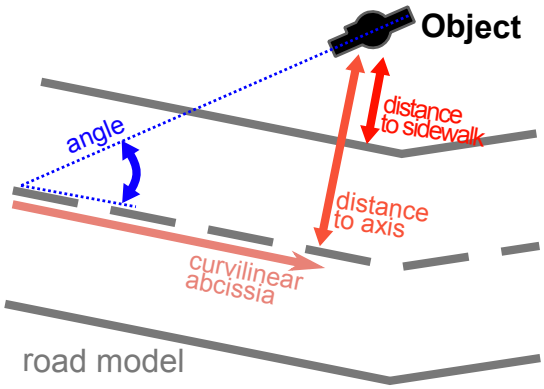


Figure 63 on page 107

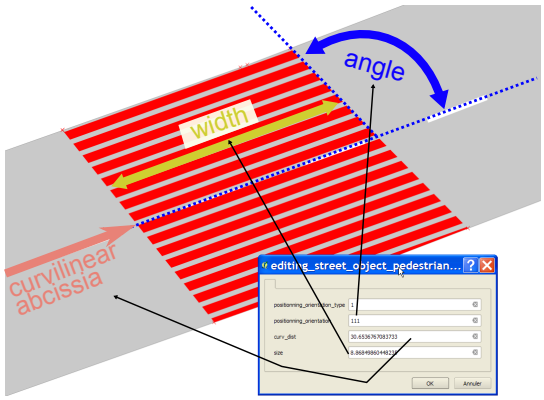


Figure 64 on page 108

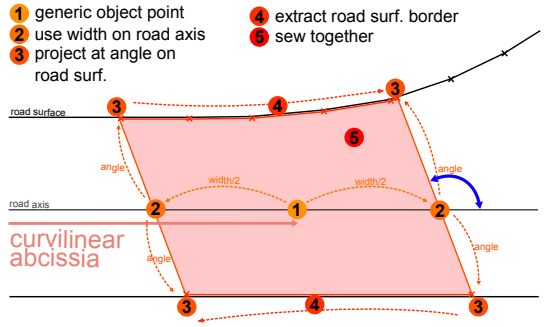


Figure 65 on page 109

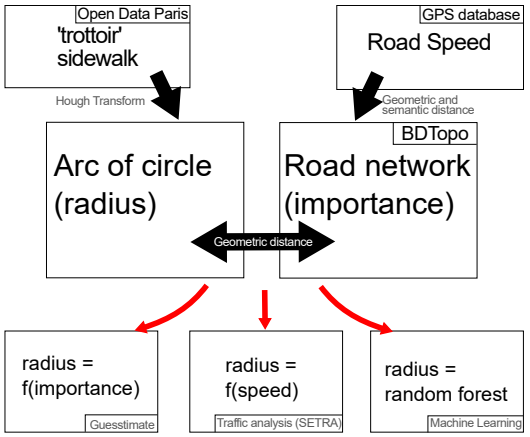


Figure 66 on page 110

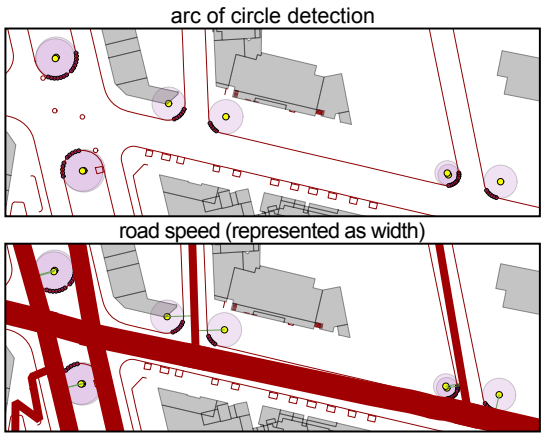


Figure 67 on page 110

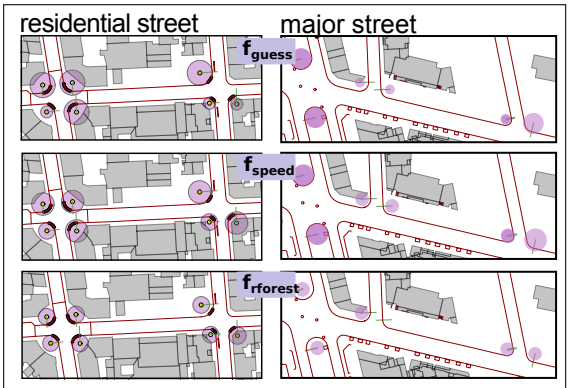


Figure 68 on page 111

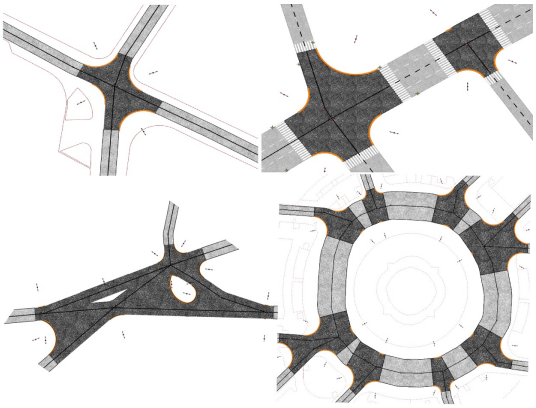


Figure 69 on page 112

Figure 70 on page 112

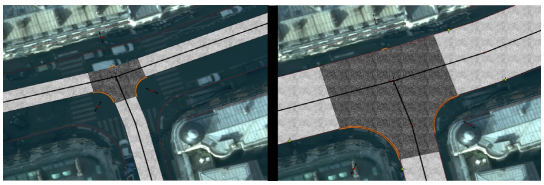


Figure 71 on page 113

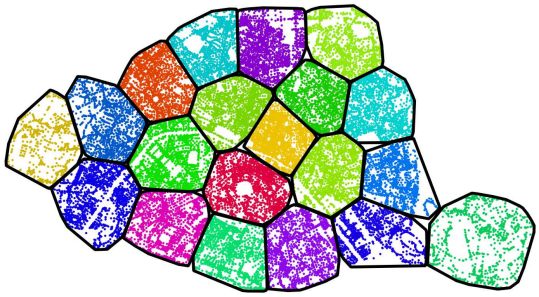


Figure 72 on page 114

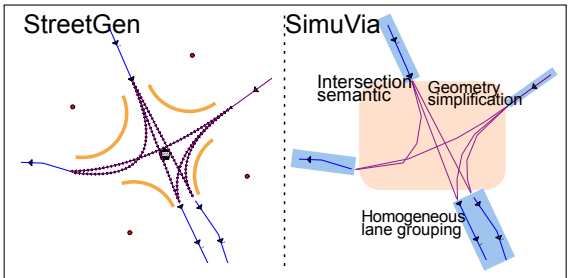


Figure 73 on page 114

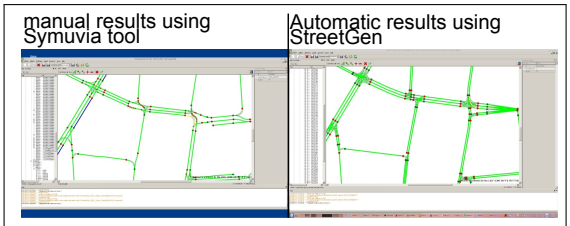


Figure 74 on page 115

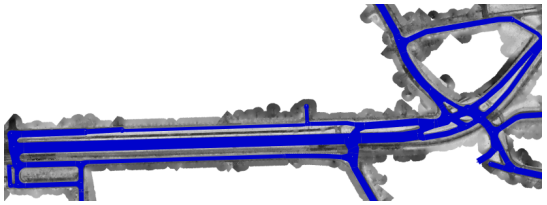


Figure 75 on page 115

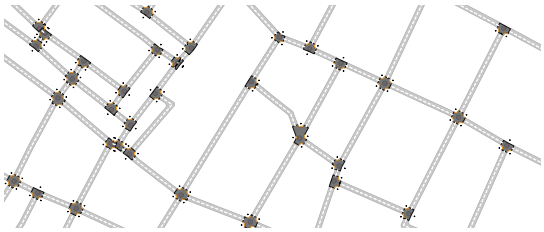


Figure 76 on page 116

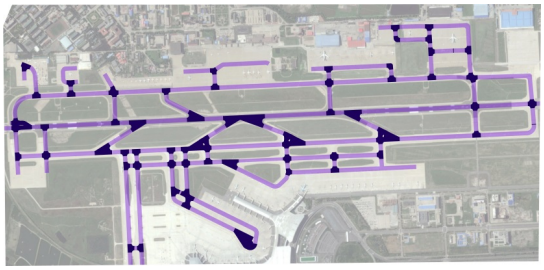


Figure 77 on page 116

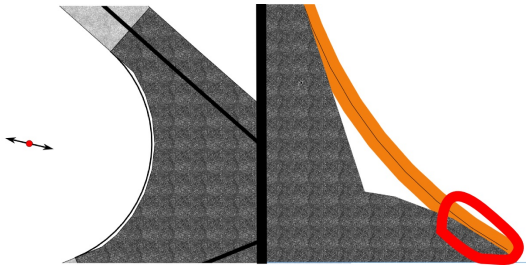


Figure 78 on page 118

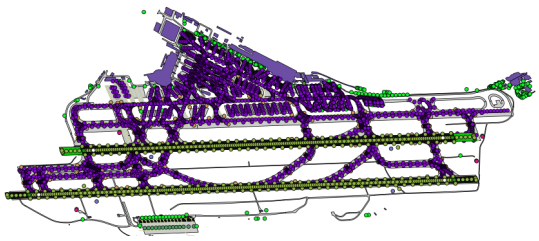


Figure 79 on page 119

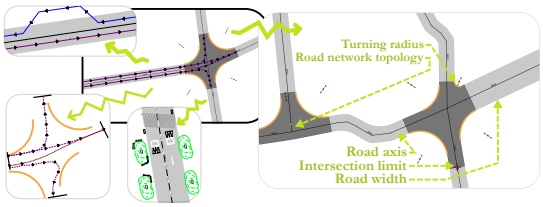


Figure 80 on page 123

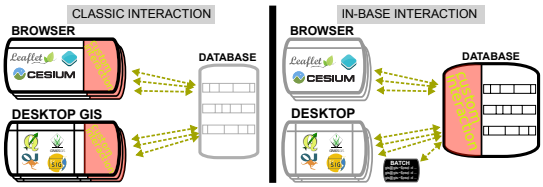


Figure 81 on page 126

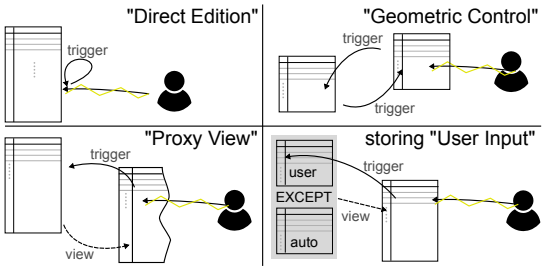


Figure 82 on page 127

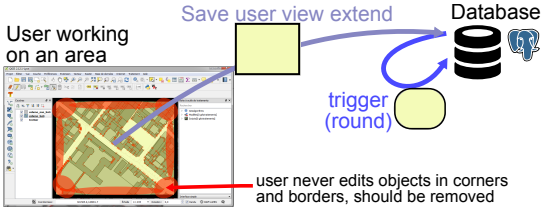


Figure 83 on page 128

video link 

Figure 84 on page 128

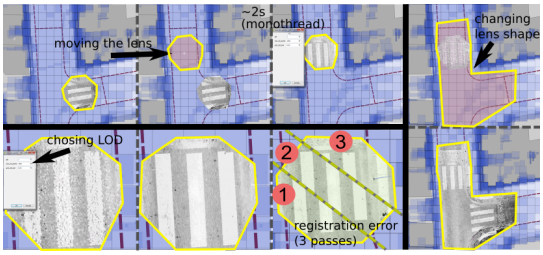


Figure 85 on page 129

video link 

Figure 86 on page 129

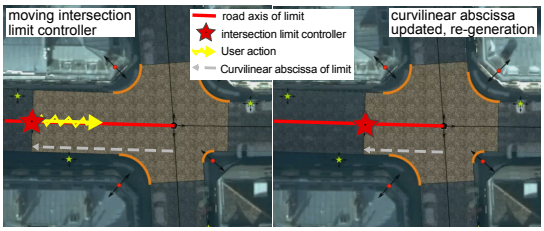


Figure 87 on page 130

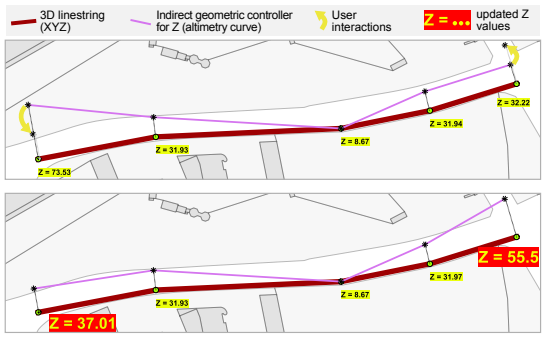


Figure 88 on page 131

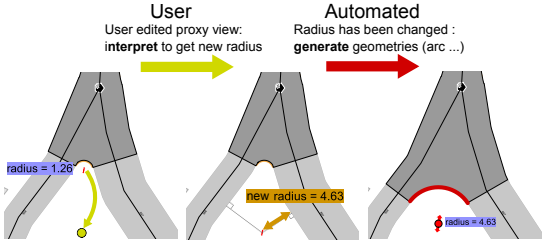


Figure 89 on page 132

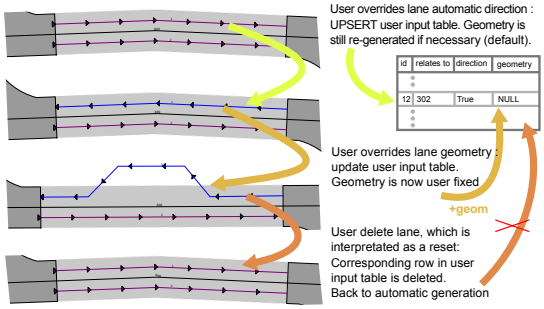


Figure 90 on page 134

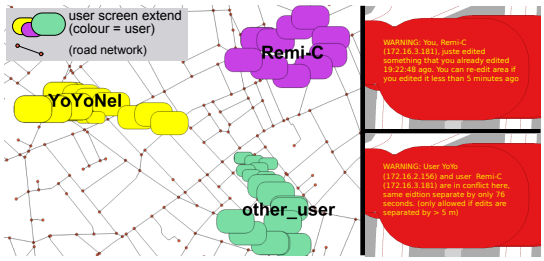


Figure 91 on page 136

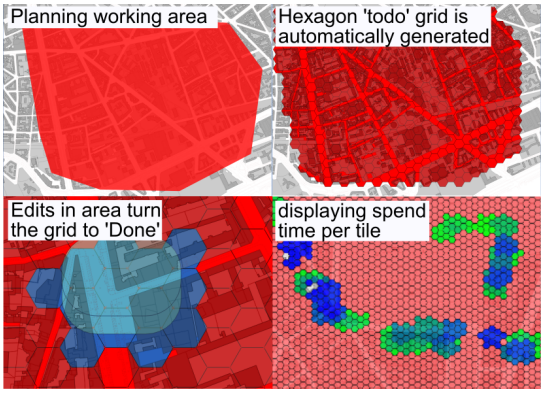


Figure 92 on page 137

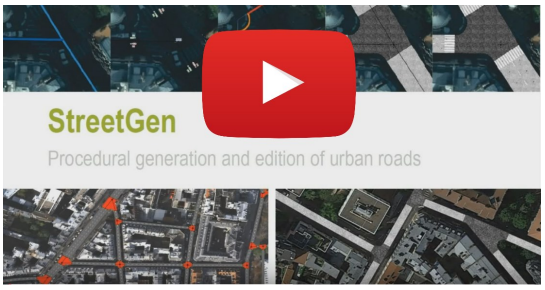


Figure 93 on page 137

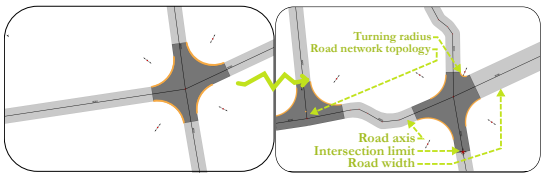


Figure 94 on page 138

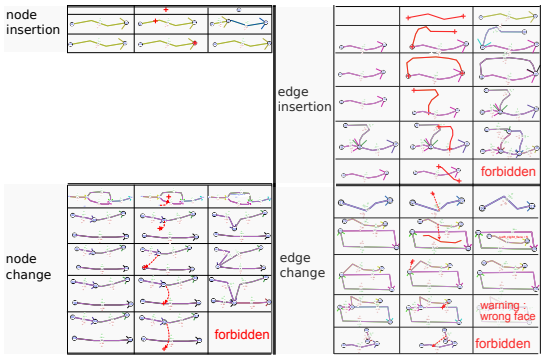


Figure 95 on page 139

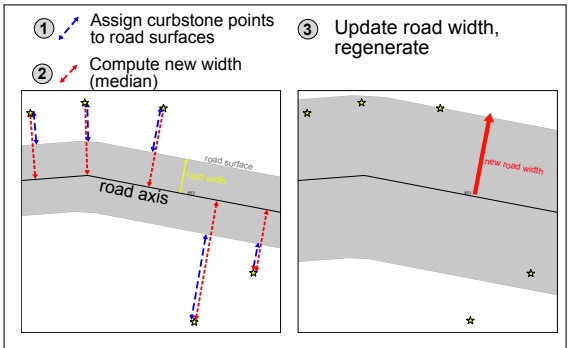


Figure 96 on page 140

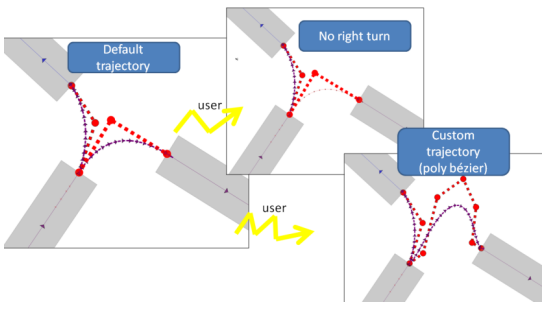


Figure 97 on page 141

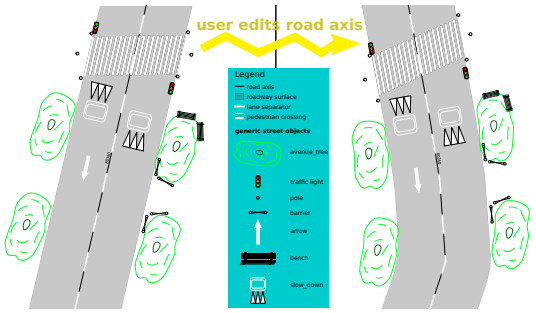


Figure 98 on page 142

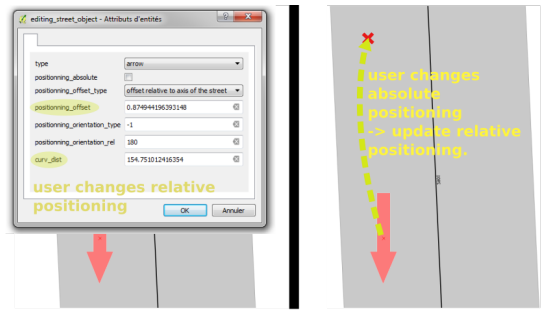


Figure 99 on page 143

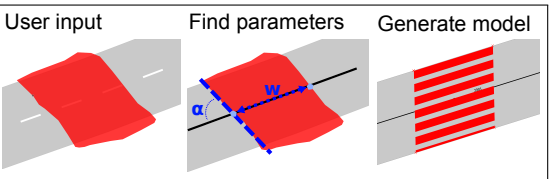


Figure 100 on page 143

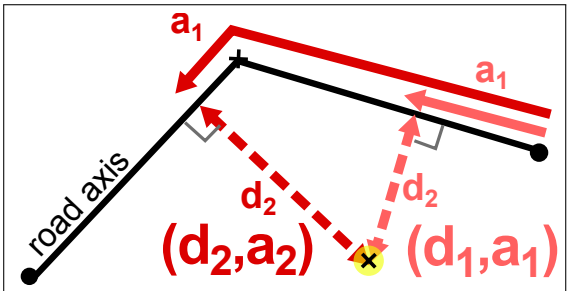


Figure 101 on page 146

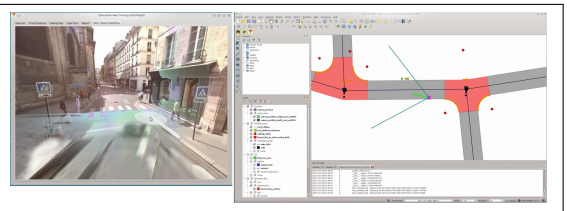


Figure 102 on page 147

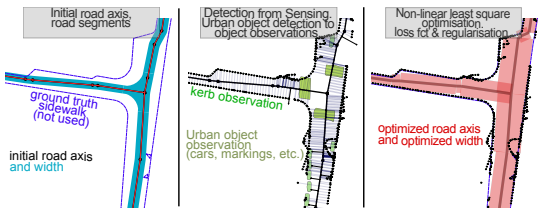


Figure 103 on page 149

Figure 104 on page 150

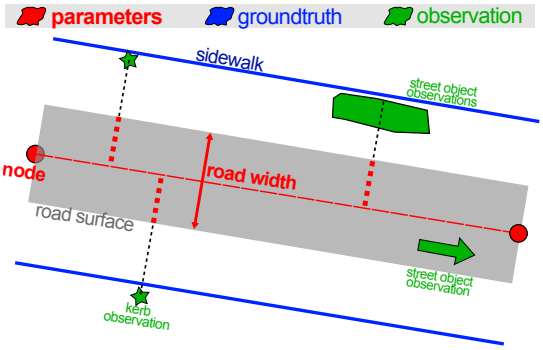


Figure 105 on page 154

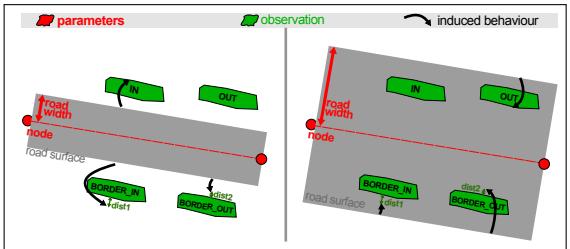


Figure 106 on page 154

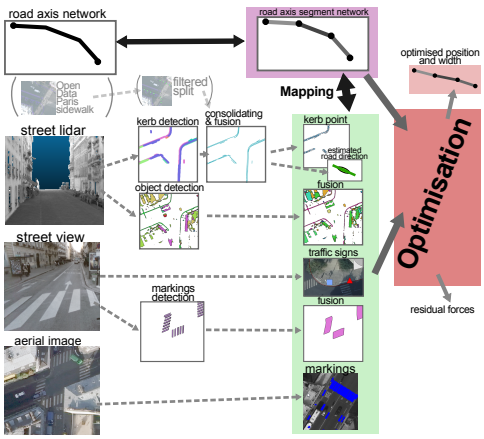


Figure 107 on page 156

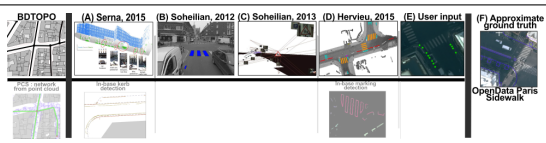


Figure 108 on page 157

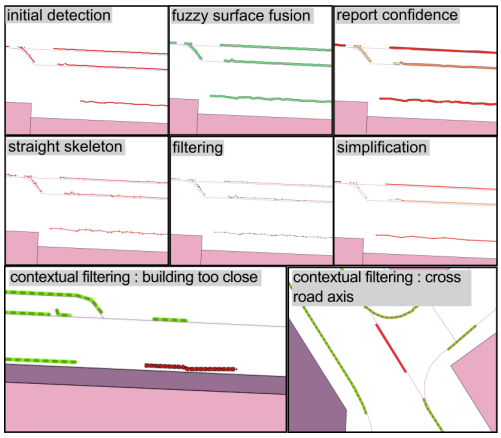


Figure 109 on page 159

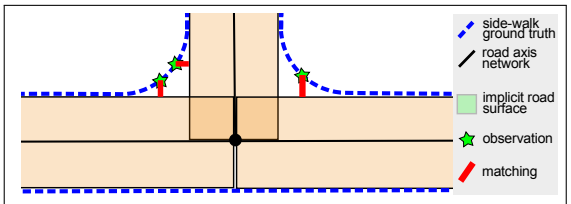


Figure 110 on page 160

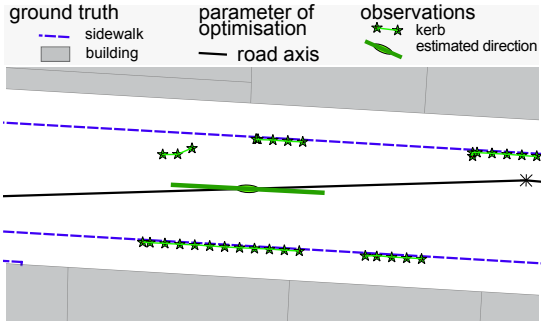


Figure 111 on page 161

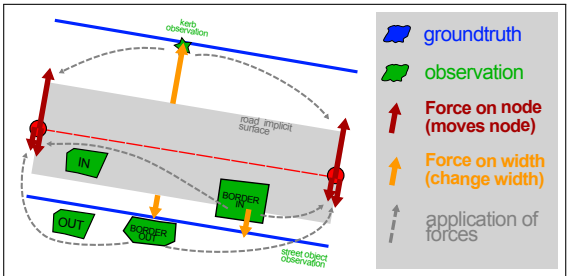


Figure 112 on page 162

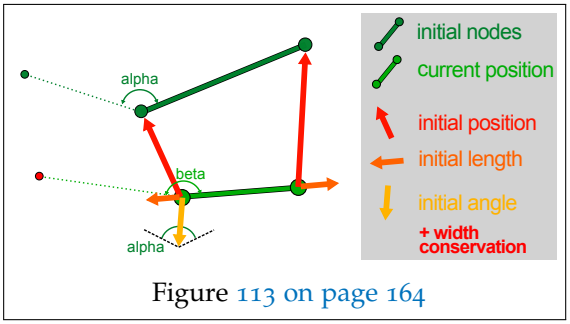


Figure 113 on page 164

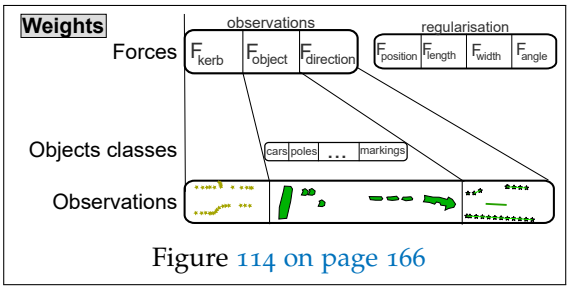


Figure 114 on page 166

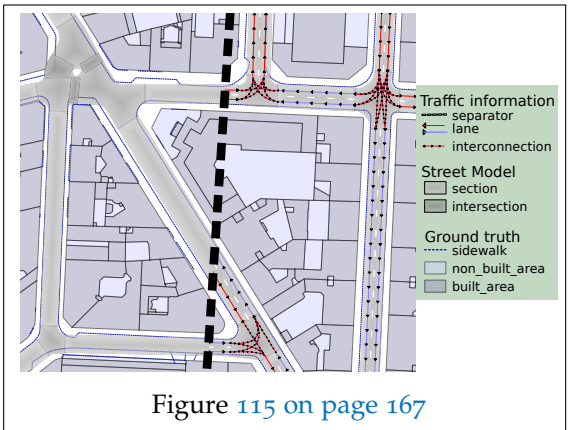


Figure 115 on page 167

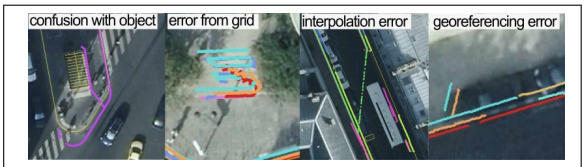


Figure 116 on page 169

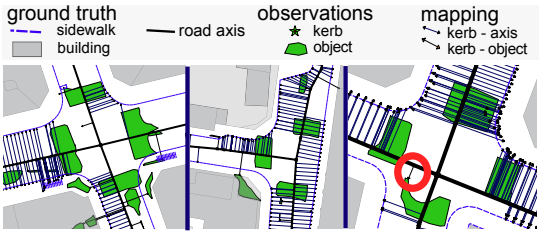


Figure 117 on page 170

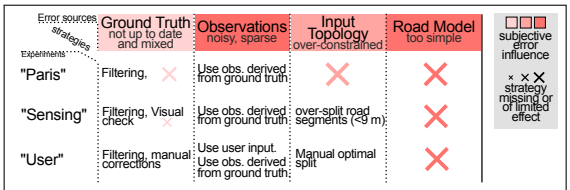


Figure 118 on page 171

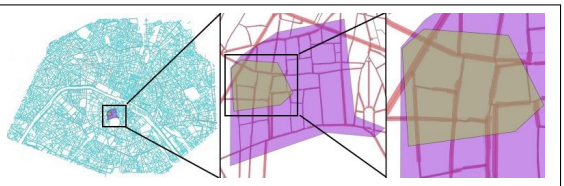


Figure 119 on page 171

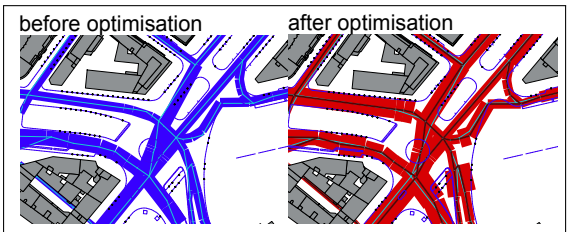


Figure 120 on page 173

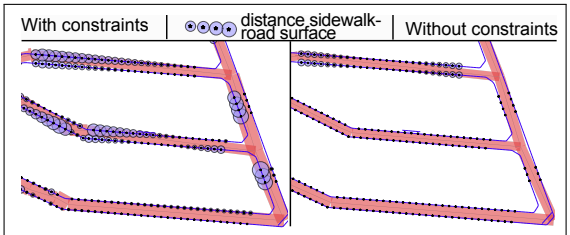


Figure 121 on page 173

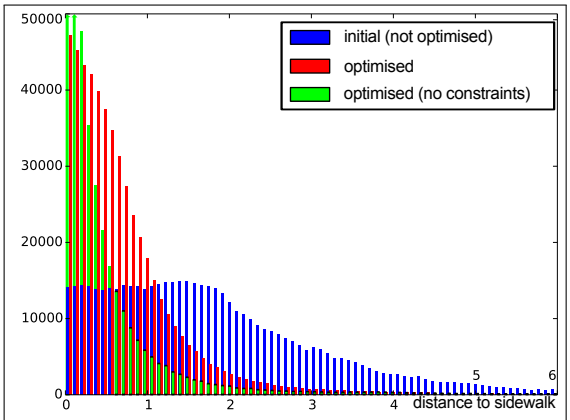


Figure 122 on page 174

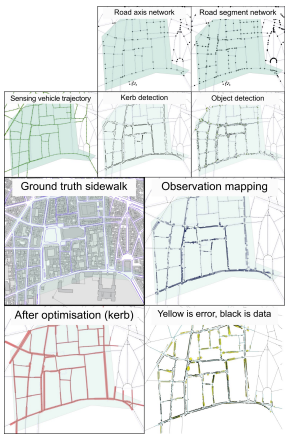


Figure 123 on page 175

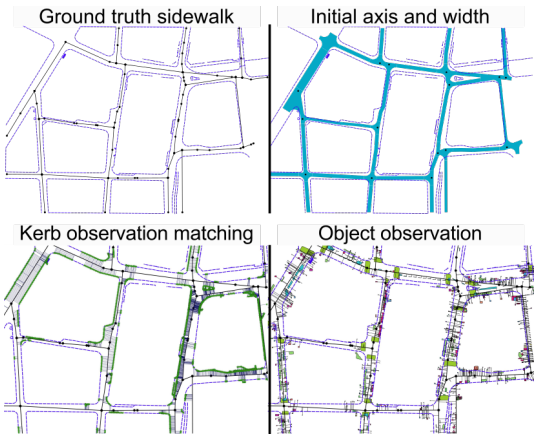


Figure 124 on page 176

Figure 125 on page 177

Figure 126 on page 178

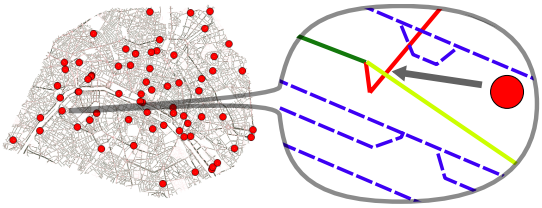


Figure 127 on page 178

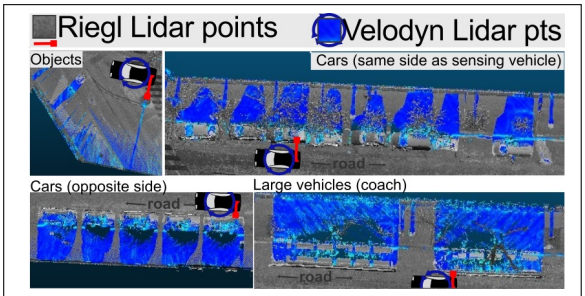


Figure 128 on page 181

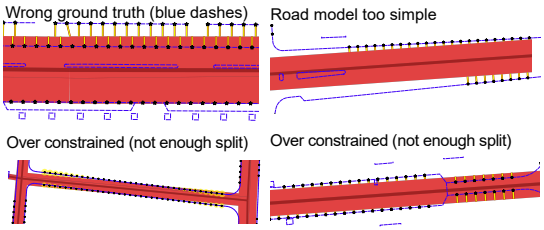


Figure 129 on page 184

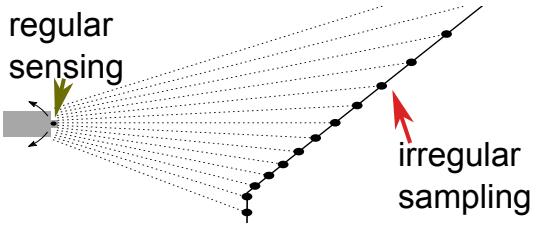


Figure 131 on page 194

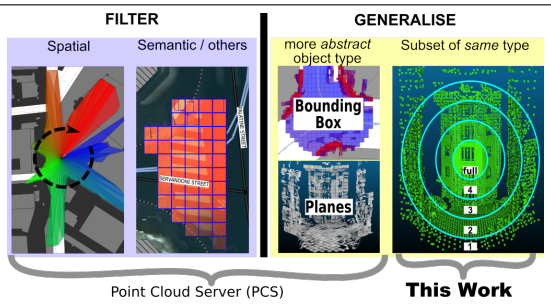


Figure 132 on page 195

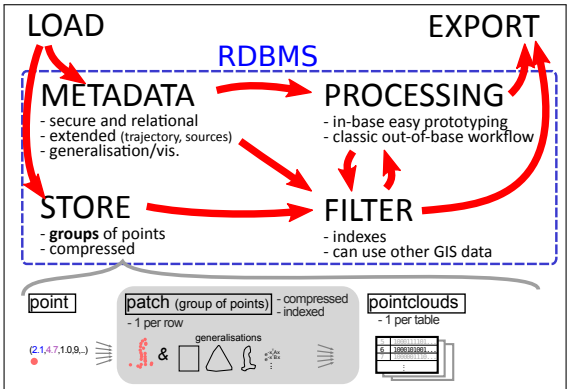


Figure 133 on page 198

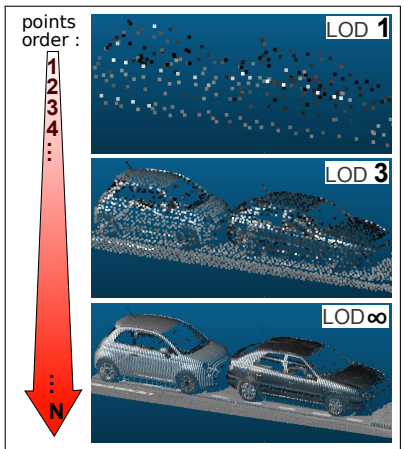


Figure 134 on page 199

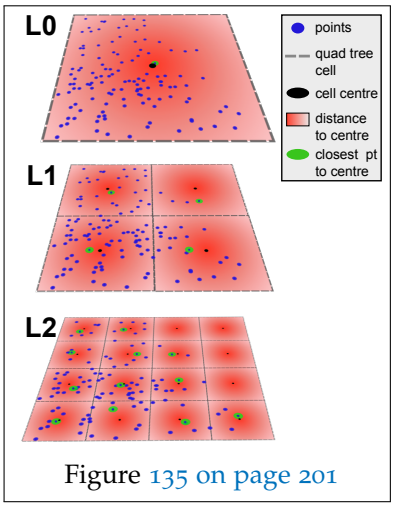


Figure 135 on page 201

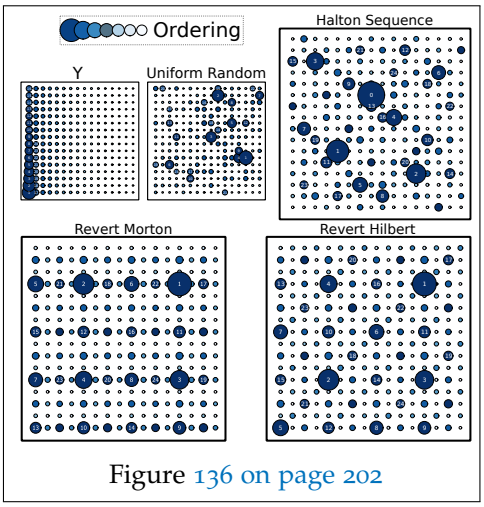


Figure 136 on page 202

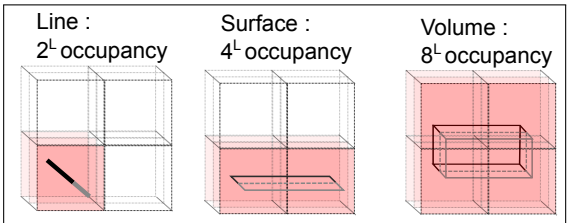


Figure 137 on page 203

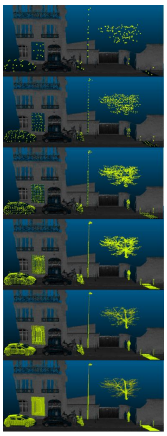


Figure 138 on page 204

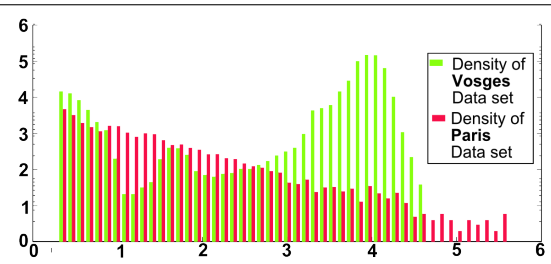


Figure 139 on page 209

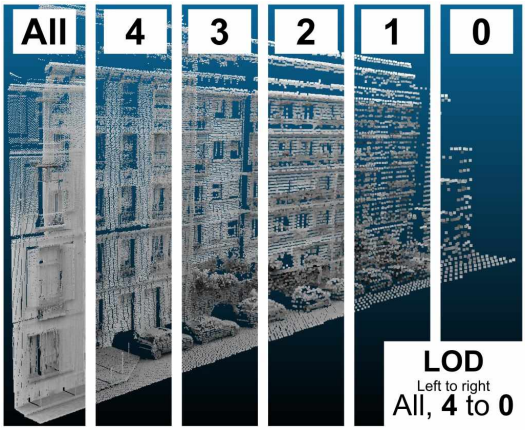


Figure 140 on page 210

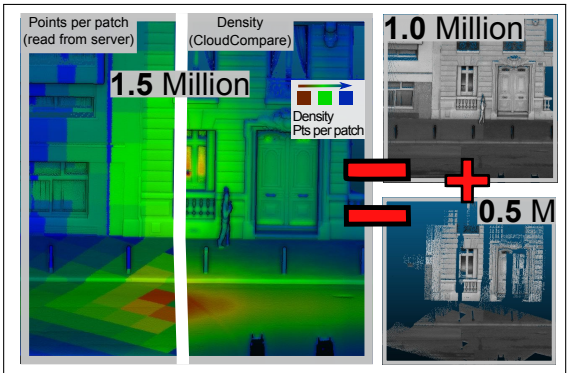


Figure 141 on page 211

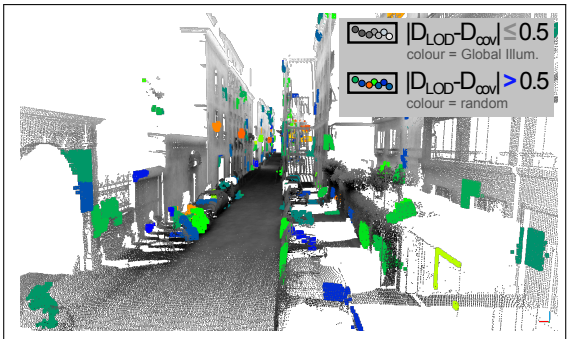


Figure 142 on page 212

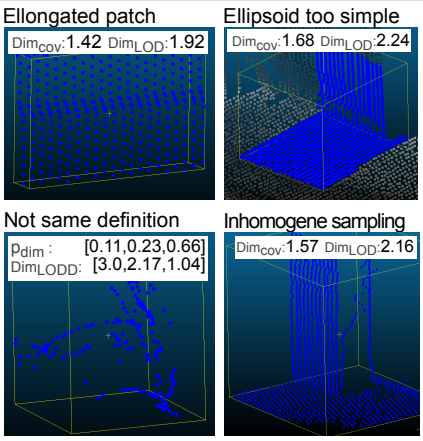


Figure 143 on page 213

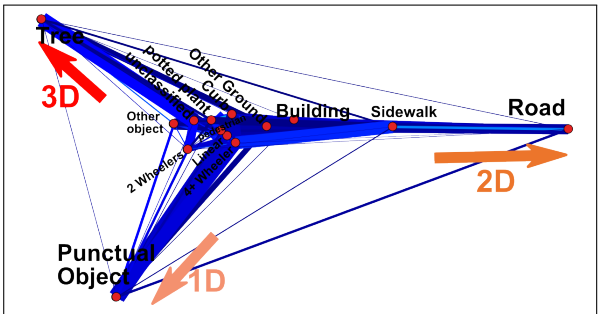


Figure 144 on page 214

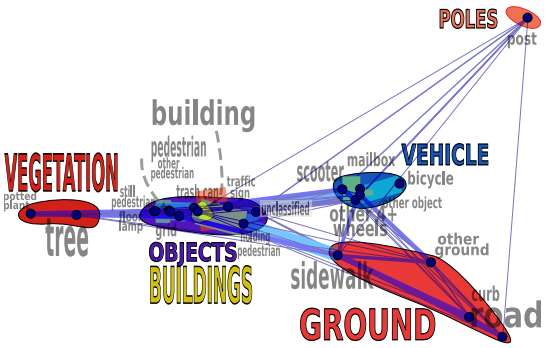
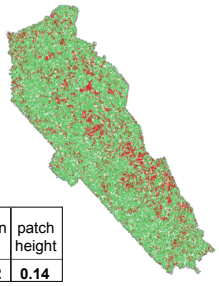


Figure 145 on page 215

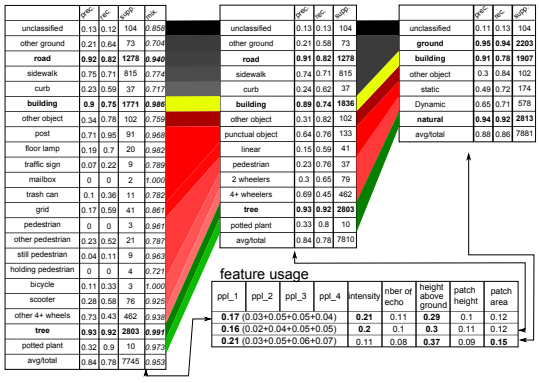
	prec.	rec.	supp.	mix.
Closed Forest	**0.99**	**0.91**	**390k**	**0.883**
Moor	0.18	0.68	8.7k	0.741
Not forest	**0.86**	**0.89**	**128k**	**1**
avg/total	0.94	0.90	526k	0.901



Feature usage

ppl_1	ppl_2	ppl_3	ppl_4	Intensity	nber of echo	mean Z	patch height
0.57 (0.07+0.13+0.18+0.19)				0.09	0.06	**0.12**	**0.14**

Figure 146 on page 216



	prec.	rec.	supp.	mix.
unclassified	0.13	0.12	104	0.858
other ground	0.21	0.64	73	0.704
road	**0.92**	**0.82**	**1278**	**0.940**
sidewalk	0.75	0.71	815	0.774
curb	0.23	0.59	37	0.717
building	**0.9**	**0.75**	**1771**	**0.986**
other object	0.34	0.78	102	0.759
post	0.71	0.95	91	0.968
floor lamp	0.19	0.7	20	0.982
traffic sign	0.07	0.22	9	0.789
mailbox	0	0	2	1.000
trash can	0.1	0.36	11	0.782
grid	0.17	0.59	41	0.861
pedestrian	0	0	3	0.961
other pedestrian	0.23	0.52	21	0.787
still pedestrian	0.04	0.11	9	0.963
holding pedestrian	0	0	4	0.721
bicycle	0.11	0.33	3	1.000
scooter	0.28	0.58	76	0.925
other 4+ wheels	0.73	0.43	462	0.938
tree	**0.93**	**0.92**	**2803**	**0.991**
potted plant	0.32	0.9	10	0.973
avg/total	0.84	0.78	7745	0.953

	prec.	rec.	supp.
unclassified	0.13	0.13	104
other ground	0.21	0.58	73
road	**0.91**	**0.82**	**1278**
sidewalk	0.74	0.71	815
curb	0.24	0.62	37
building	**0.89**	**0.74**	**1836**
other object	0.31	0.82	102
punctual object	0.64	0.76	133
linear	0.15	0.59	41
pedestrian	0.23	0.76	37
2 wheelers	0.3	0.65	79
4+ wheelers	0.69	0.45	462
tree	**0.93**	**0.92**	**2803**
potted plant	0.33	0.8	10
avg/total	0.84	0.78	7810

	prec.	rec.	supp.
unclassified	0.11	0.13	104
ground	**0.95**	**0.94**	**2203**
building	**0.91**	**0.78**	**1907**
other object	0.3	0.84	102
static	0.49	0.72	174
Dynamic	0.65	0.71	578
natural	**0.94**	**0.92**	**2813**
avg/total	0.88	0.86	7881

feature usage

ppl_1	ppl_2	ppl_3	ppl_4	intensity	nber of echo	height above ground	patch height	patch area
0.17 (0.03+0.05+0.05+0.04)				**0.21**	0.11	**0.29**	0.1	0.12
0.16 (0.02+0.04+0.05+0.05)				**0.2**	0.1	**0.3**	0.11	0.12
0.21 (0.03+0.05+0.06+0.07)				0.11	0.08	**0.37**	0.09	**0.15**

Figure 147 on page 216

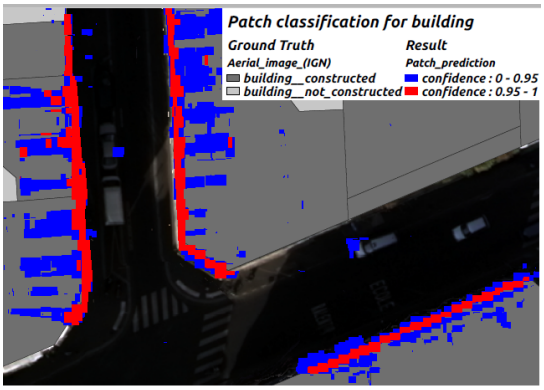


Figure 148 on page 217

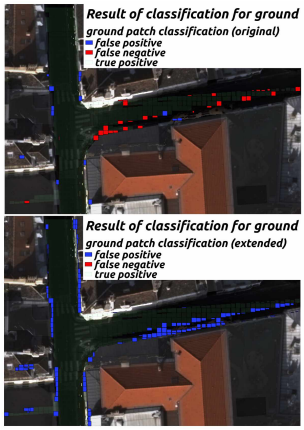


Figure 149 on page 218

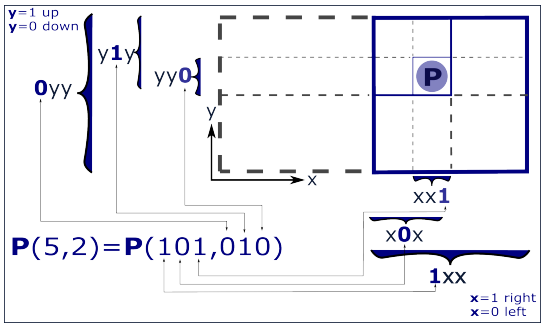


Figure 150 on page 220

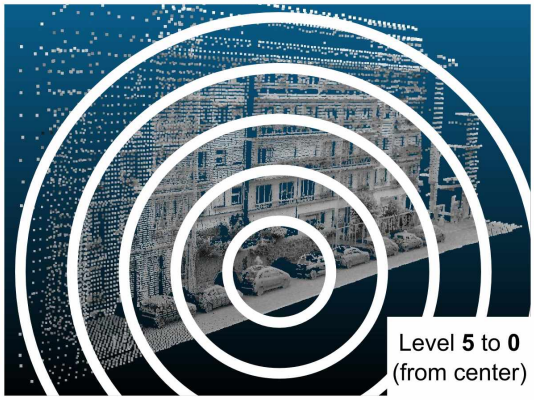


Figure 151 on page 221

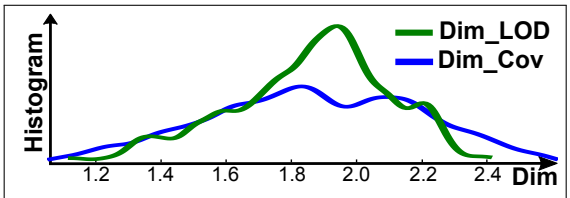


Figure 152 on page 221

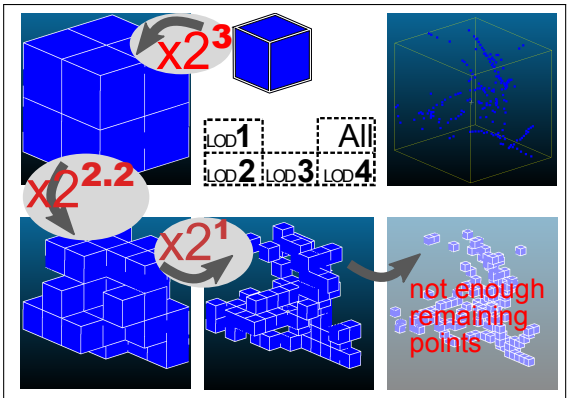


Figure 153 on page 222

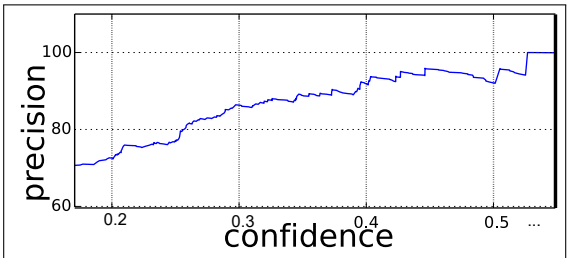


Figure 154 on page 224

ABSTRACT

World urban population is growing fast, and so are cities, inducing an urgent need for city planning and management. Increasing amounts of data are required as cities are becoming larger, "Smarter", and as more related applications necessitate those data (planning, virtual tourism, traffic simulation, etc.). Data related to cities then become larger and are integrated into more complex city model.

Roads and streets are an essential part of the city, being the interface between public and private space, and between urban usages. Modelling streets (or street reconstruction) is difficult because streets can be very different from each other (in layout, functions, morphology) and contain widely varying urban features (furniture, markings, traffic signs), at different scales.

In this thesis, we propose an automatic and semi-automatic framework to model and reconstruct streets using the *inverse procedural modelling* paradigm. The main guiding principle is to generate a procedural generic model and then to adapt it to reality using observations. In our framework, a "best guess" road model is first generated from very little information (road axis network and associated attributes), that is available in most of national databases. This road model is then fitted to observations by combining in-base interactive user edition (using common GIS software as graphical interface) with semi-automated optimisation. The optimisation approach adapts the road model so it fits observations of urban features extracted from diverse sensing data. Both street generation (StreetGen) and interactions happen in a database server, as well as the management of large amount of street Lidar data (sensing data) as the observations using a Point Cloud Server.

We test our methods on the entire Paris city, whose streets are generated in a few minutes, can be edited interactively (<0.3 s) by several concurrent users. Automatic fitting (few min) shows promising results (average distance to ground truth reduced from 2.0m to 0.5m).

In the future, this method could be mixed with others dedicated to reconstruction of buildings, vegetation, etc., so an affordable, precise, and up to date City model can be obtained quickly and semi-automatically. This will also allow to such models to be used in other application areas. Indeed, the possibility to have common, more generic, city models is an important challenge given the cost an complexity of their construction.