ENSG Géomatique

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

National School of Geographics Sciences

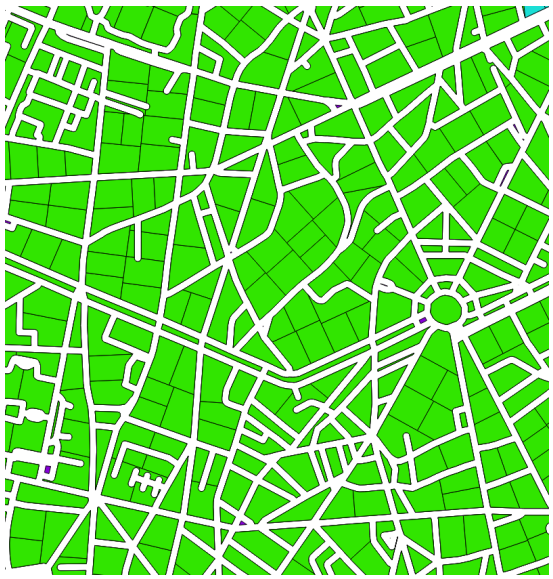
AUCIGEN

Automated City Generation

AuCiGen Project Report

Advanced procedural City Generation

Different implementations of realistic procedural parcel splitting



Groupe de projet n° 4

Arnaud Grégoire, Salim Alioua, Houssem Adouni, El-Hadi Bouchaour

☒ Non confidentiel ☐ Confidentiel IGN ☐ Confidentiel Industrie ☐ Jusqu'au ...

National School of Geographics Sciences
6-8 Avenue Blaise Pascal - Cité Descartes - 77420 Champs-sur-Marne
Téléphone 01 64 15 31 00 Télécopie 01 64 15 31 07

Contents

Glossary & abreviations

OBB École Nationale des Sciences Géographiques

GDAL Geospatial Data Abstraction Library

CGAL Computational Geometry Algorithms Library:is a software project that provides easy access to efficient and reliable geometric algorithms in the form of a C++ library. CGAL is used in various areas needing geometric computation, such as geographic information systems, computer aided design, molecular biology, medical imaging, computer graphics, and robotics.[1]

Introduction

AuCiGen (Automated City Generation) is a global project which is composed of 5 subproject. Our team, Arnaud Grégoire (Product Owner), Houssem Adouni (Scrum Master), El-Hadi Bouchaour (developper) and Salim Alioua (developper) were assigned on the part : "Procedural Generation of Parcels in Urban Modelling". During 4 weeks, we worked together following the Scrum way of development with the goal of implementing two different algorithm for realistic procedural parcel splitting.
We realised a total of 4 sprints (each sprint has a duration of one week) with more than 30 user stories.

The first sprint, we focused our work in the installation, configuration and comprehension of two major libraries :

- CGAL and its dependencies (Boost, Gmp, Mpfr, Cmake) in order to compute the skeleton of a polygon
- GDAL / OGR for other spatial analysis computing

The second sprint, the team focused its work on two tasks :

- the implemetation of the oriented bouding box based algorithm
- the implementation of the straight skeleton based algorithm

The third sprint was following the previous one, continuing the development of both codes.
During the last sprint, we started closing tasks of projects :

- Completing documentation of code, using Doxygen
- Coding Unit test using Catch2 framework
- Writing Latex report using Overleaf
- Preparing the presentation powerpoint

0.1 Input parameters project

The previous group was charged to transform the network of linestrings generated by group 1&2 into a shapefile of polygons modelising realistic road/streets network like presented on the figure 1a listed below.
However, our project subject was to do realistic subdivision of parcels. That's why we had to do some transformations to create parcels from given roads network.

- Firstly, we merged all roads segment into one huge polygon
- Secondly, we filled areas that were enclosed by roads
- Next we deleted parcels with an area bigger than 1 square meters. This allow us to prevent parcels from being built on what are green space areas, rivers or big construction like train station.

The figure 1b listed below presents the input data that we created after those transformations listed above. We can see that parcels have been correctly created following the street network. Besides, The train station "Gare d'Austerlitz", the "Jardin Des Plantes" green space area and the river "La Seine" has been not detected as parcels like we intended. However, we can see that the portion of river enclosed by the bridge "Pont d'Austerlitz" and bridge "Pont Charles de Gaulle" has been detected as parcels. In fact, the area enclosed in not enough big to be considered as an area to delete.

(a) Roads buffers

(b) Parcels

Figure 1: Input project datas

0.2 Spatial analysis processing

Our entire project is based on the article : "Procedural Generation of Parcels in Urban Modeling" written by Carlos A.Vanegas, Tom Kelly & al. [5]. This scientific publication go deeply in details in parcels splitting and describe two differents approach based on two different concept.

- The first approach used the oriented bounding box of a polygon in order to split by half of it width. This method will be explained in the first chapter.
- The second approach is based on the straight skeleton of a polygon. Along this skeleton, interior buffers with a determined offset will be apply and parcels will be created by slicing those buffers along street directions. This method will be detailled in the second chapter.

0.3 Output data

The output data of our project consist in a shapefile containing splitted parcels under a given threshold. This parcels are represented as OGRPolygon Object (same as wkbPolygon).
As our projects requires two very different approach of procedural generation of parcels, we decided to implement both of them separatly.

Figure 2: Example of computed parcels splitted with the OBB method

Chapter 1 Oriented bounding box subdivision algorithm

1.1 Polygons associated geometry

1.1.1 Convex hull

"In mathematics, the convex hull or convex envelope of a set X of points is the smallest convex set that contains X. For instance, if X is a wkbPolygon, the convex hull may be visualized as the shape enclosed by a rubber band stretched around X." [2]
The definition can be applied on our type of geometry : OGRPolygon (wkbPolygon). To compute convex hull of each polygons, we used the OGR implementation of convex hull computation :

```
1 OGRGeometry * OGRGeometry::ConvexHull () const
```

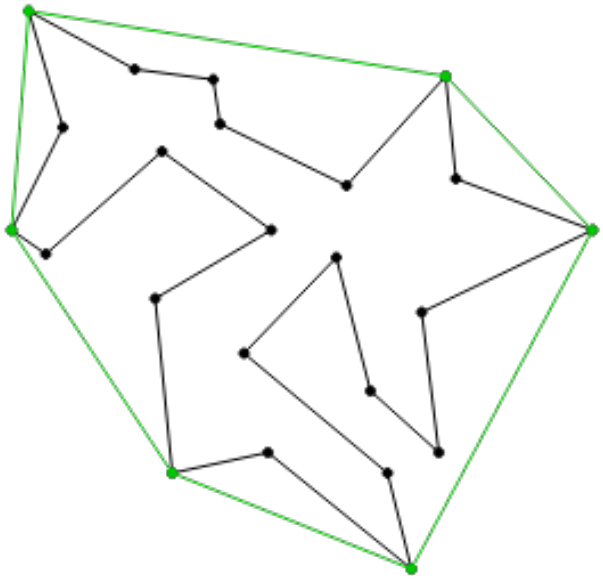


Figure 1.1: Convex Hull computed by OGR [3]

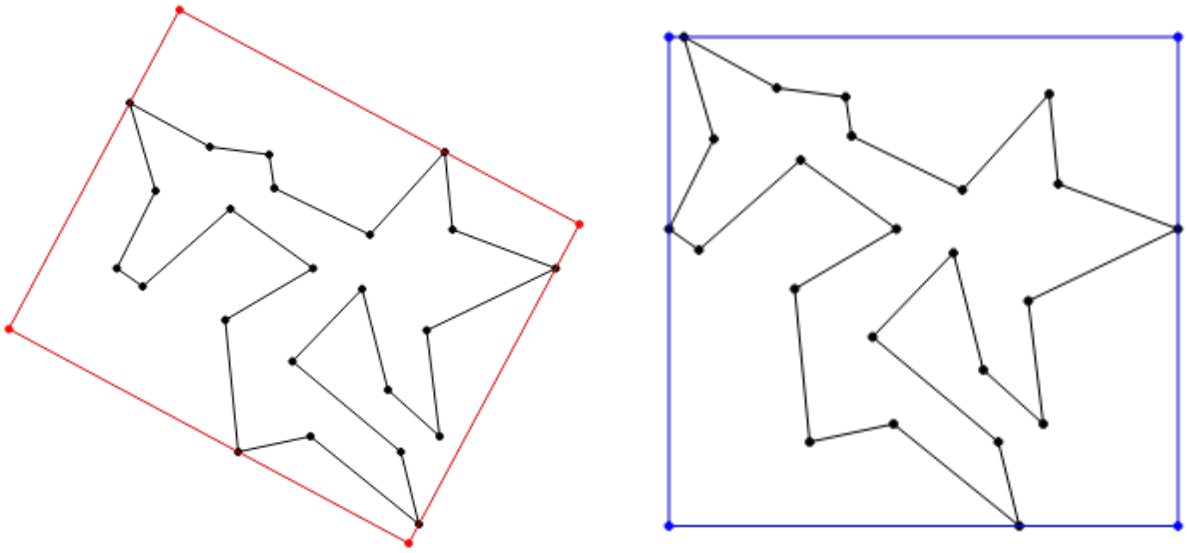
The above figure show the convex hull (in green) of a input polygon (in black). All points of the convex hull are part of the input polygon. The convex hull has particular properties that will be used to compute the minimum oriented bounding box.

1.1.2 Bounding box

There are differents possibles bounding box. As shown in the figure listed below, the easiest computable box is the axis-aligned one 1.2a. This one is given by the OGR method

```
1 void OGRGeometry::getEnvelope (OGREnvelope * psEnvelope) const
```

An other type of bounding box is the minimum oriented one. Unfortunately, There is no computation of the minimum oriented bounding box in OGR/GDAL. We had to implement a version by ourselves.



(a) Minimum oriented bounding box (b) Axis-aligned bounding box

Figure 1.2: Different types of bounding box [3]

1.2 Concept : Rotating Calipers

Everything coded in that part is a re-implementation in C++ of the JavaScript computation of minimum oriented bounding box coded by David Geidav [3]. His JavaScript code is available at : `https://github.com/geidav/ombb-rotating-calipers`

1.2.1 Mathematical Algorithm theory

The common algorithm used for computed the minimum oriented bounding box is based on a concept : the rotating calipers.

The concept of the algorithm is based on a theorem that say :
" *smallest-area enclosing rectangle of a polygon has a side collinear with one of the edges of its convex hull.*"[3]

Thanks to this theorem the number of minimum oriented bounding box candidates is dramatically reduced to the number of convex hull edges. The complexity of the algorithm is although reduced to N (linear).

1.2.2 Algorithm Concept

The implemented algorithm follows those instructions [3]:

- We compute the convex hull of our polygon
- We find the extreme points of the convex hull
- We construct two vertical lines and two horizontal that bound our polygon
- We initialize our area this bound's area
- We rotate that rectangle until one of its edge is colinear to an edge of the convex hull
- We compute the area of this new rectangle
- If this area is this candidate is smaller than the previous one, we update the minimum area and stock this candidate as a possible minimum oriented bounding box

- We repeat this step until all edges of the convex hull is colinear with one of our bounds.

The figure listed below presents oriented bounding box candidates computed by our C++ reimplementation of JavaScript rotating calipers.

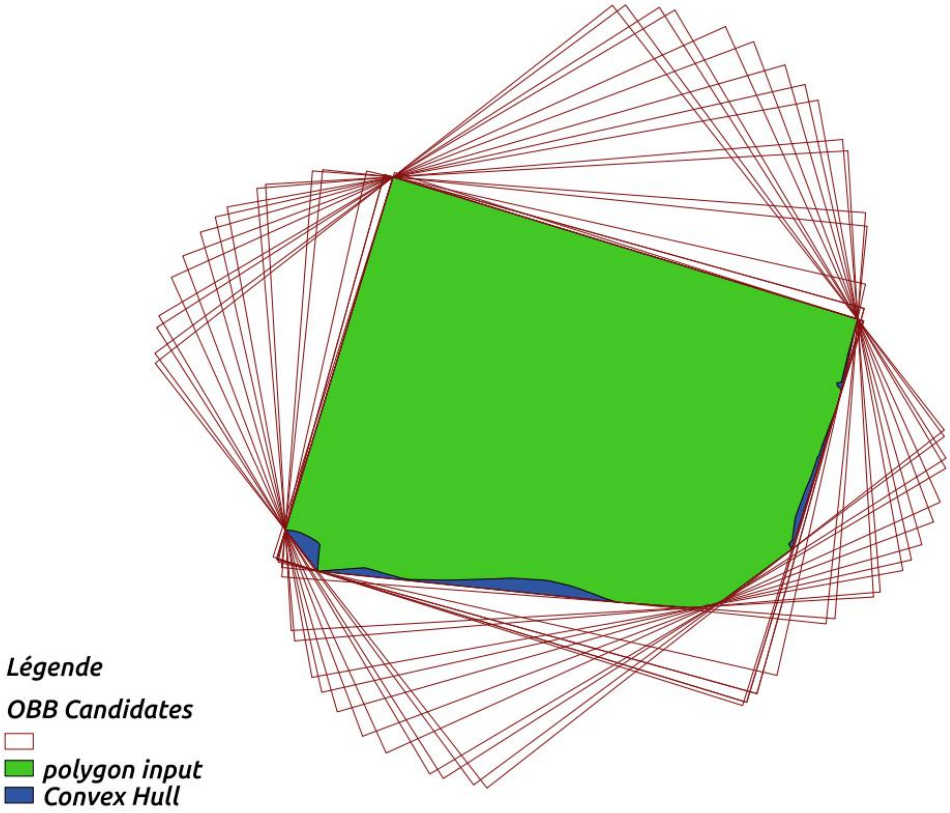


Figure 1.3: Oriented bounding box candidates

The number of convex hull edges is always equal to the number of Oriented bounding box candidates. It is one of the weakness of the rotating calipers algorithm. In fact, if there is a circle-shaped polygon, he will be composed by a huge amount of points that will greatly extend the time spent in finding the minimum oriented bounding box.

1.3 Different global splitting methods

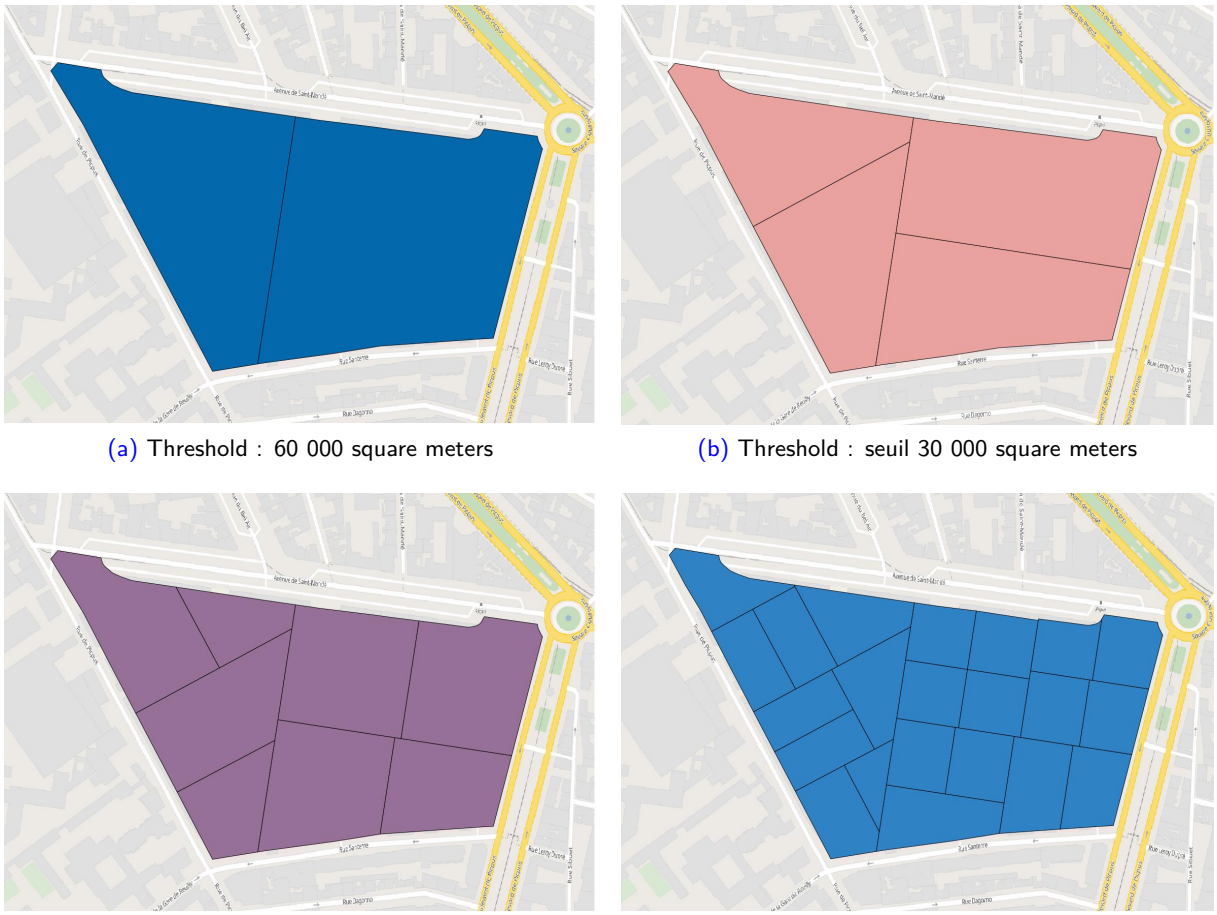
The computation of the minimum oriented bounding box is used to split polygons along the the OBB width or height.

1.3.1 Split along the OBB width

They could be two types of split in function of two parameters, the height and the witdh of the minimum oriented bounding box.

- If the width is bigger than the height, the algorithm vertically sliced the polygon along the height at the half width.

- If the height is bigger than the width, the algorithm horizontally sliced the polygon along the width at the half height.



(a) Threshold : 60 000 square meters

(b) Threshold : seuil 30 000 square meters

(c) Threshold : seuil 15 000 square meters

(d) Threshold : 6 000 square meters

Figure 1.4: Split of a polygon using OBB method with different thresholds

Figures listed above show us polygons splitted made by the minimum oriented bounding box algorithm. We can see that the more the area decrease, the more the number of subdivisions increase. Besides, we observe experimentally that under a certain threshold, the oriented bounding box method creates rectangle parcels very similar to their neighbour.

1.3.2 Imperative or recursive method

The splitting method of a polygon has been implemented into both ways : the imperative one and the recursive one. Both are returning the same results, the only difference is the time spent by each method. Experimentally, we compared time spent by different methods in function of threshold's area.

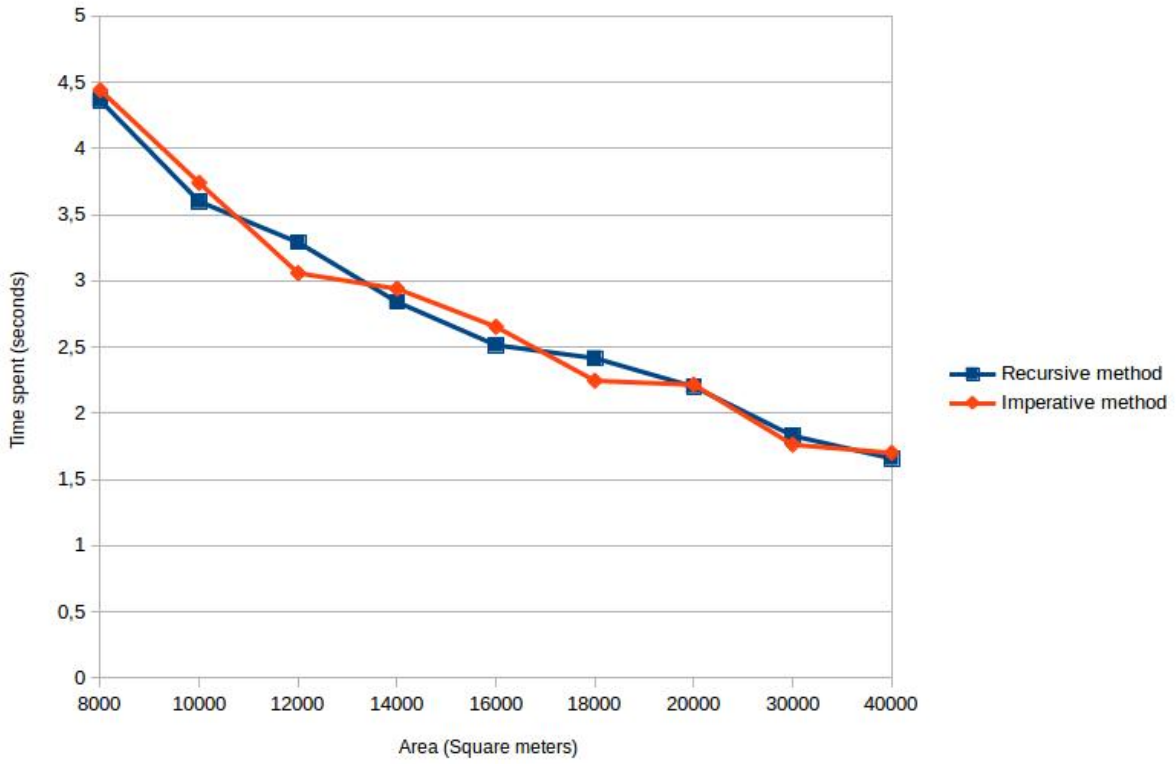


Figure 1.5: Computation time of recursive and imperative method in function of threshold (area)

The figure 1.5 listed above represents the time spent by each method to calculate and write splitted polygons. We can see that there is no significative difference between the two implementations. One explication coud be that the time to read and write polygons in shapefile is the major cause of computation time.

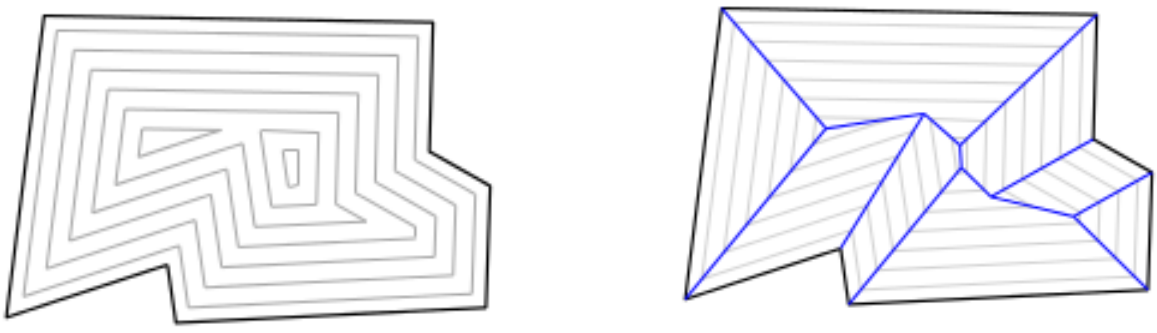
Chapter 2 Straight skeleton subdivision algorithm

2.1 Objective

Our work consist of performing a partitioning of the interior of the city blocks based on user specified subdivision attributes and styles parameter's using the straight skeleton algorithm which means the Subdivision of the space in between roads (Blocks) into parcels.

2.2 Straight skeleton definition

"The straight skeleton of a polygon is defined by a continuous shrinking process in which the edges of the polygon are moved inwards parallel to themselves at a constant speed. As the edges move in this way, the vertices where pairs of edges meet also move, at speeds that depend on the angle of the vertex. During the shrinking process, arcs are created by following the vertices of the shrinking polygon. A node is created when the shrinking process requires combinatorial or topological changes of the polygon. A combinatorial change is required when an edge vanishes. When the polygon is split into two parts, a topological change of the polygon occurs. The straight skeleton partitions the polygon into cells. If one of these moving vertices collides with a nonadjacent edge, the polygon is split in two by the collision, and the process continues in each part. The straight skeleton is the set of curves traced out by the moving vertices in this process." In the illustration the top figure shows the shrinking process and the middle figure depicts the straight skeleton in blue. [4]



(a) Offset Polygons (b) Straight skeleton

Figure 2.1: Straight skeleton subdivision treatment

2.3 Fields and domains applications

There are several applications of the staright skeleton ,we cite some examples as follows:

- Construction of a polygonal roof over a set of ground walls.

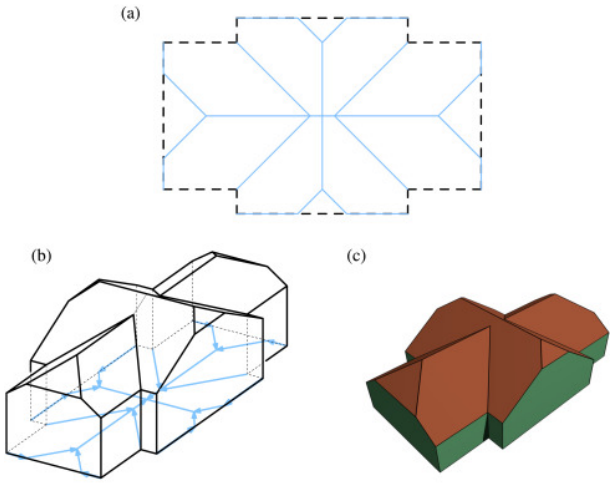


Figure 2.2: example of roof construction
[4]

- Reconstruct a geographical terrain from a river map.

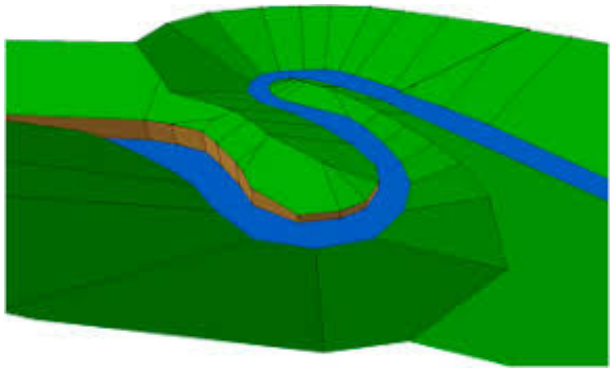


Figure 2.3: example of construction of geographical terrain
[4]

- Study of impact of rain water fall on floodings.

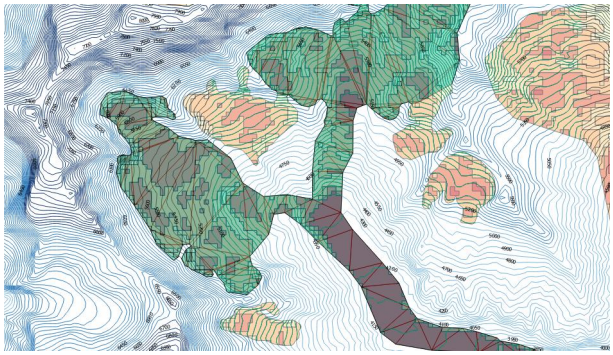


Figure 2.4: using straight skeleton in study of impact rain water
[4]

2.4 Algorithm steps implementation

To implement straight skeleton algorithm, we proceed to doing the following steps:

2.4.1 Documentation:

In order to understand straight skeleton algorithm, it was essential to read and consult many reference sites internet and documentations about the processus of this algorithm,in addition to other dependencies library such as CGAL,Boost,Gmp,Mpfr.

2.4.2 Preparation of the environment:

- Operating System: Ubuntu
- Progrmming language : C++ (Codebooks EDI)
- SIG Tools: QGIS to display input and output shapefiles vector format;
- GDAL/OGR library: to read from and write on the shapefiles;
- CGAL and its dependences libraries (GMP,BOOST...): Calculate straight skeleton and creation offset polygons;

2.4.3 Code implementation:

- Open the input shapefile (output shapefile of third group) and extract its different features using GDAL library;

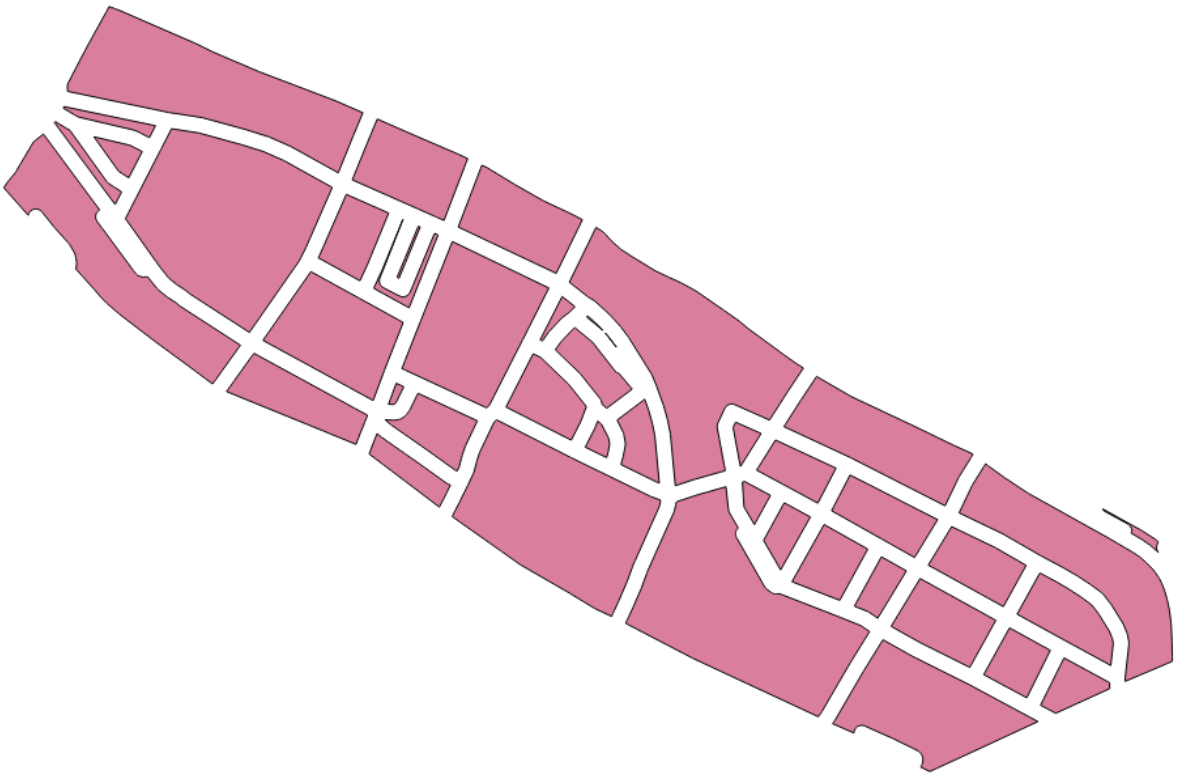


Figure 2.5: Input shapefile example
[4]

This figure represents a set of polygons that we got it after we extracted the parcels from the road Netwok shapefile.

- Compute the straight Skeleton of the different polygons (features extracted).

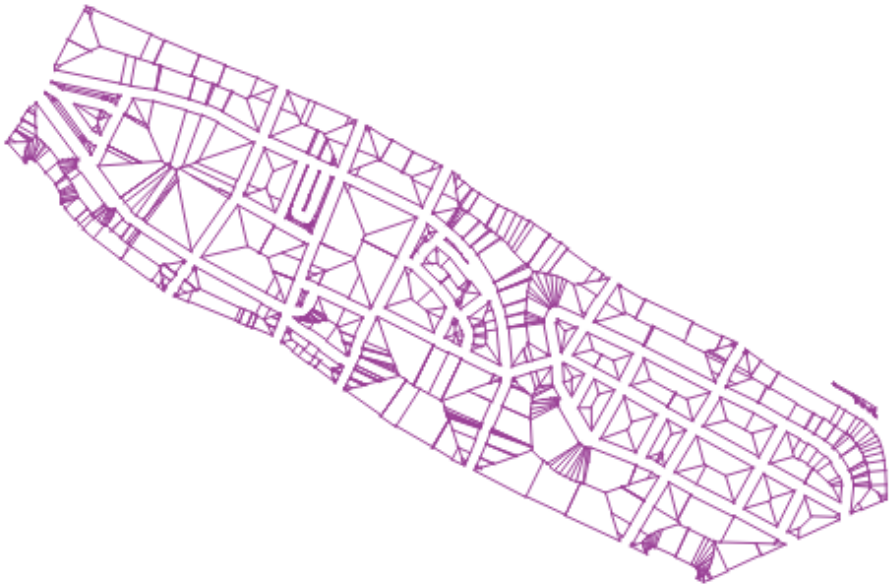


Figure 2.6: example-compute skeleton of the input shapefile polygon [4]

In the example above we used the CGAL Library to compute skeleton of each polygon of the inupt shapefile.

- Create the interior offset polygon of straight skeletons using CGAL library: in this step we apllied the offset polygon on each polygon of our shapefile.

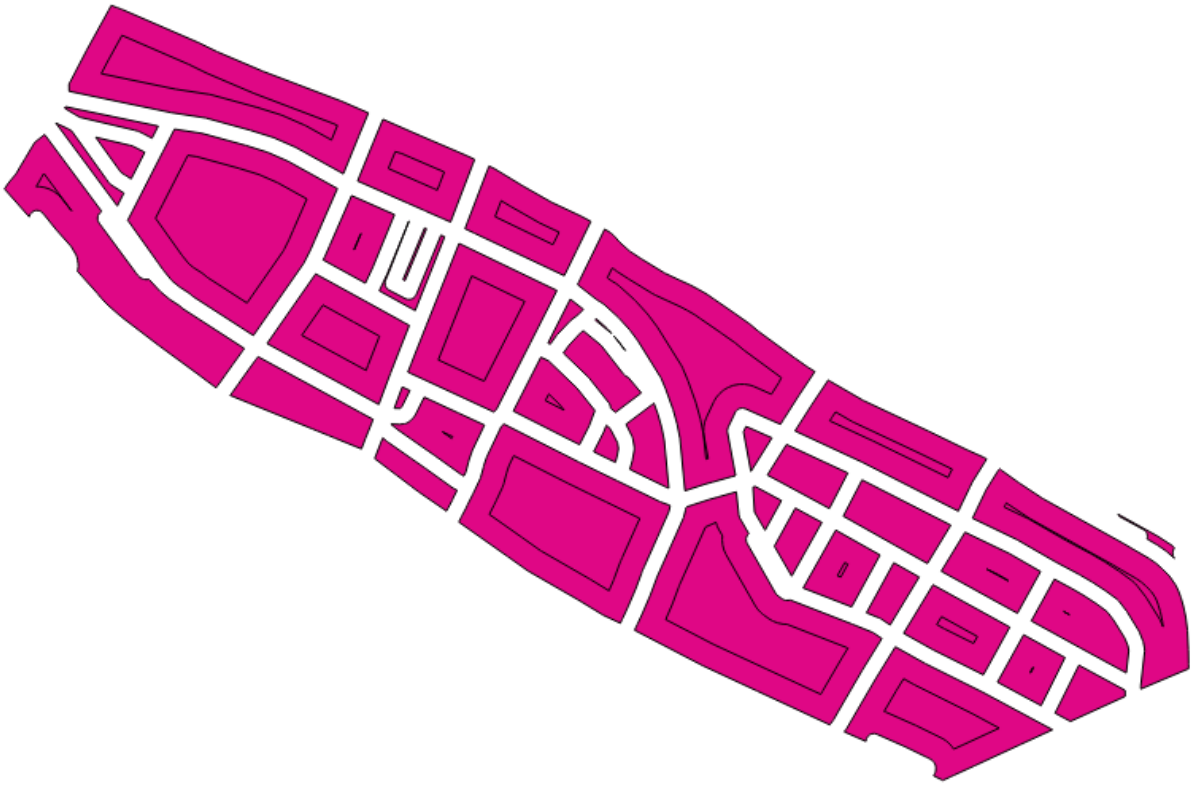


Figure 2.7: example- skeleton offset polygon (feature)
[4]

- Extract strips: this extraction was carried out through several operations namely
  a- Extract faces generated after skeleton computing.

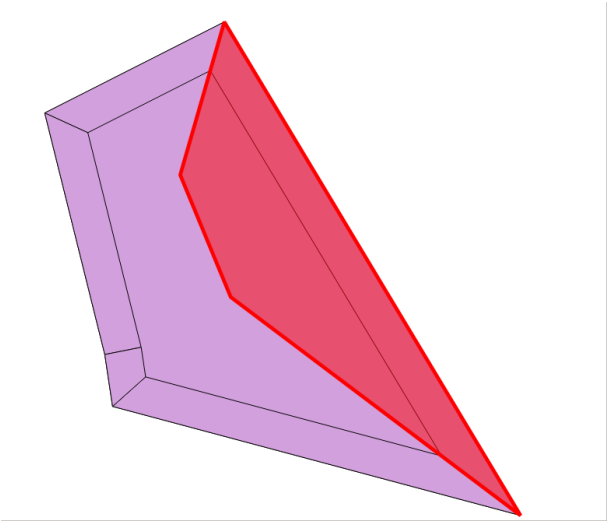


Figure 2.8: example- Faces extraction
[4]

b- Intersect the offset polygon and each of faces result of the prévious opération to have a set of polygons intersections

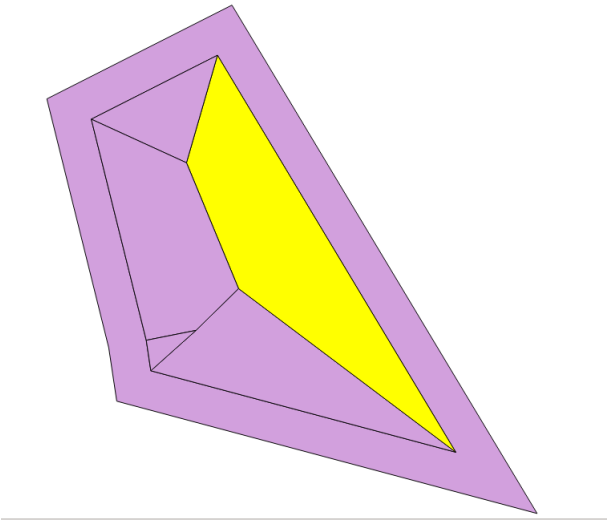


Figure 2.9: example- Faces extraction
[4]

c- Extract the difference between each face and the result of the previous step.
The result of third operation constitute the strips.

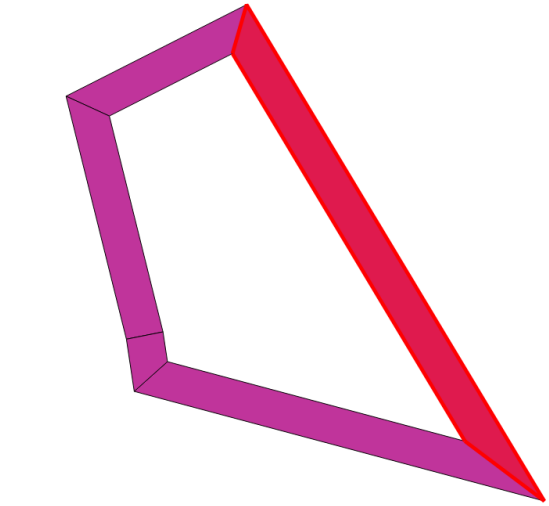


Figure 2.10: example- Faces extraction
[4]

‘

- subdivsion of stips into slices (parcels): in this step we used the same technique of the first algorithm (OBB) to divide the stripes into parcels (the smallest bounding rectangle).

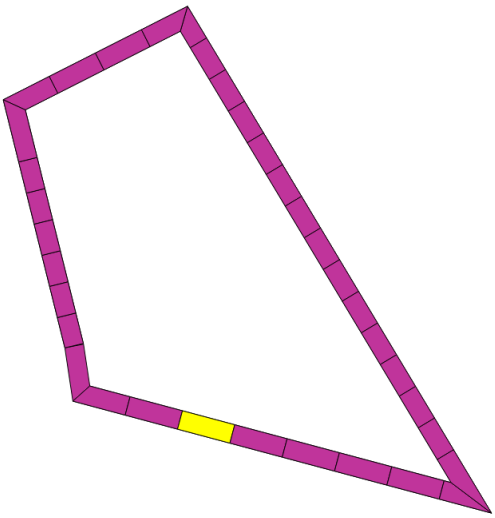


Figure 2.11: example- subdividing strips into slices (Parcels)
[4]

- Correction of slices on the edge's diagonal;

2.4.4 encountered difficulties

some of the difficult problems that we encountered during the algorithm implementation, are cited below:

- the installation and the configuration steps of the CGAL library and its dependencies libraries ,were a little bit difficult due to the lack of documentation related to this topic.
- we found many problems with code blocks IDE,in many cases it crashes suddenly which caused a lot wasted of time.

Conclusion

During 4 weeks, we worked together following the Scrum way of development. Our team, learned a lot about procedural parcels generation, and achieved to implements two algorithm :

- the first one based on oriented bounding box
- the second focused on straight skeleton offset

Our approachs perform a partitionning of the interior of parcels using those two algorithm. This allow any users of our project to transform a polygon-shaped street network into a partition of subdivided parcels enclosed by given streets.
Nevertheless, as our work is just a part of the main project AuCiGen, we had to integrate our work with previous (Creation of polygon-shaped street from linestring network) and next project (3D textured and procedural generation of building).

In order to ensure the global functionment of AuCiGen, we adapt our input (One parcel polygon) to the output of the project 3 (Multipolygon-shaped street network).

Besides, we went sure to get along with group 5, by modifying our output, to make the adaptation for group 5 between our output and their input.

Bibliography

[1] *CGAL definition*.

[2] *Convex Hull Definition*. Nov. 2017.

[3] David Geidav. "Computing oriented minimum bounding boxes in 2D". In: (Apr. 2014).

[4] Lazlo Szirmay-Kalos. "Proceedings of Spring Conference on Computer Graphics". In: (Nov. 2017).

[5] Carlos Vanegas. "Procedural Generation of Parcels in Urban Modeling". In: (Apr. 2012).