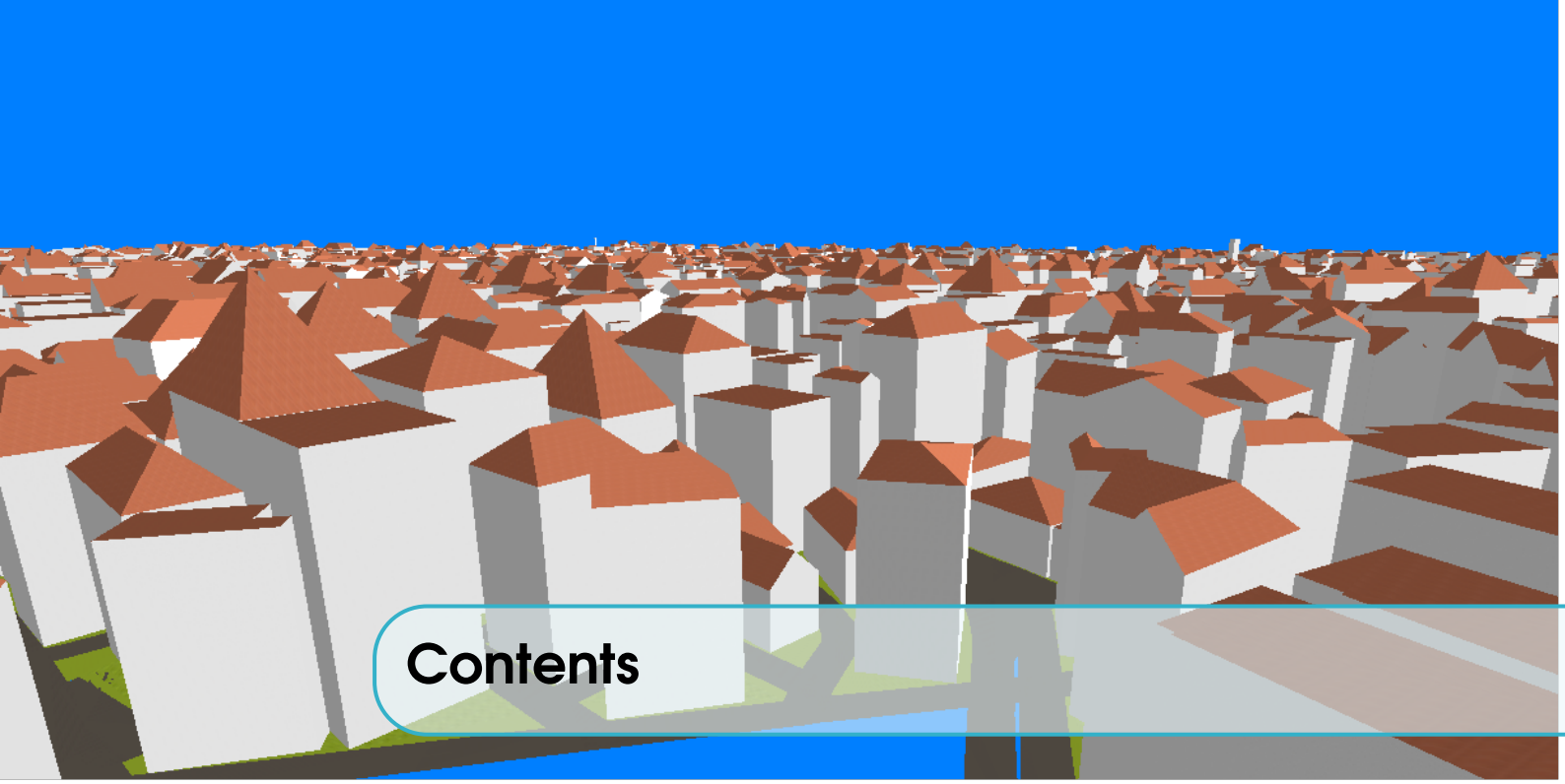
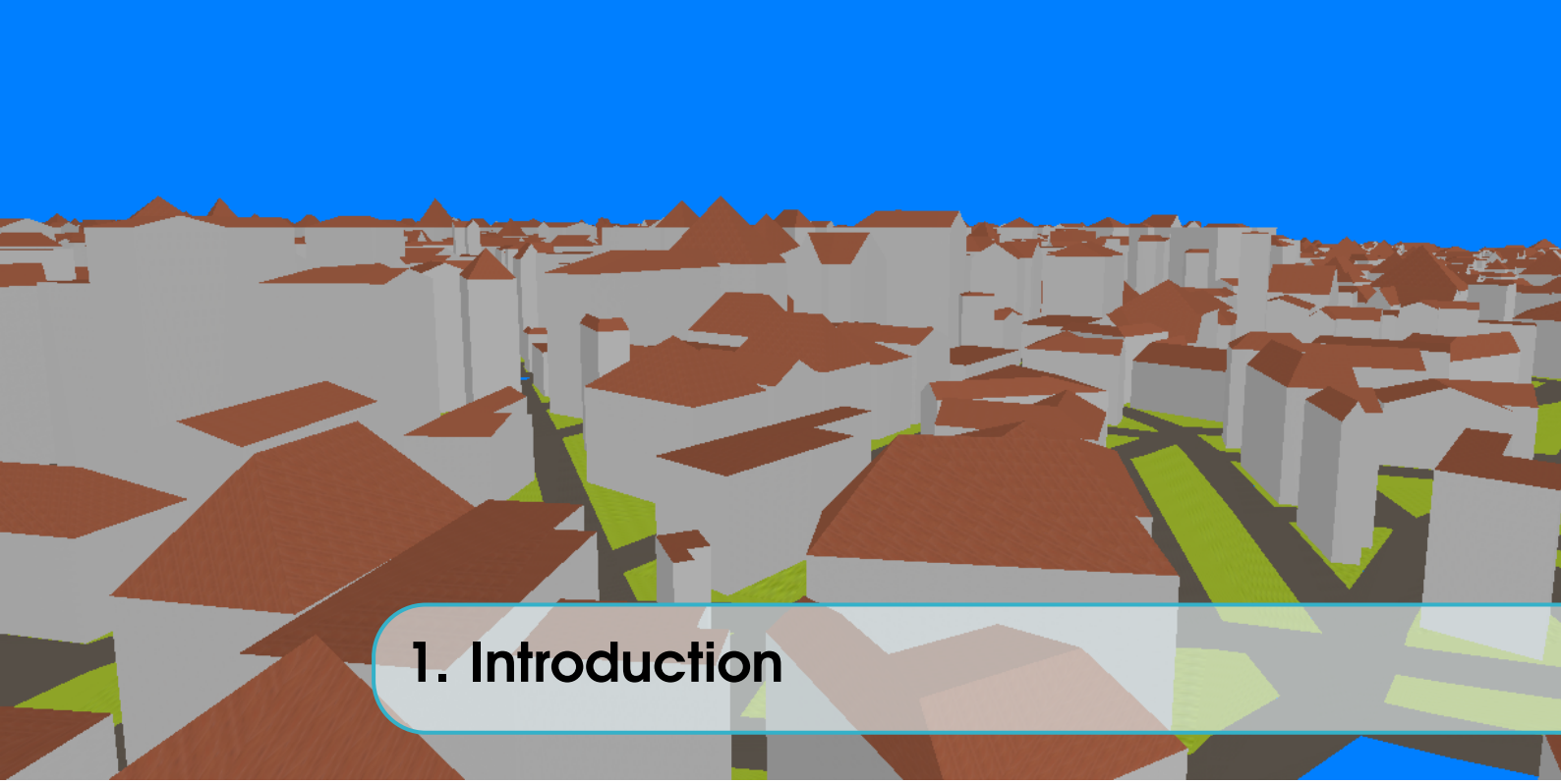
ABuGeR Automated Building Generation and Rendering

AuCiGen - Automated City Generation

Amaury Zarzelli,
Antoine Moutou,
Ismaël Esseddik,
Ludivine Schlegel



Contents



1. Introduction

1.1 Description of the subject

The geomatic project of the TSI class of 2017 consists in creating a program that generates procedurally a city in three dimensions. The name of this project is ***AuCiGen*** (Automated City Generation). the project is divided into five sub-projects:

- subject 1: creation of main roads (line),
- subject 2: creation of secondary roads (line),
- subject 3: transformation of lines into polygons,
- subject 4: creation of the parcels,
- subject 5: creation of the buildings and visualization.

Our team is composed of 4 members and worked on the subject 5. Our sub-project name is ABuGeR, for Automated Building Generation and Rendering. With a shapfile of roads and a shapfile of parcels as inputs, we create the buildings over the parcels and the 3D model of roads, parcels and buildings in Wavefront *.obj* format.

1.2 Articles abstract

1.2.1 Interactive Geometric Simulation of 4D Cities (2009)

In section 5.2 and 5.3 of the article, a set of rules to determine the buildings' height based on profitability is described. Here are the main variables introduced.

- **Building envelope**: The length between the lot limit and the building limit depends on the type of the lot and the neighboring lots' type: the distance between two buildings and the distance between the building front and the street are limited.
- **Lot price** :

```
lot_price= area * city_m2_price * lot_value / (SUM_i lot_value[i]/number_lot)
```

- **Number of floors**:

```
lot_floor = lot_floor_space / lot_envelope_area
```

- **Building orientation**: The front side is set on the side of the biggest street around the building.

- **Required floor space**: The effective area in square meters where residents live and work that would make the building profitable.

```
lot_floor_space = lot_price*margin(type)
```

with margin(type): A truncated Gaussian distribution defined per land use type.

1.2.2 Procedural Modeling of Buildings (2006)

This article is about "CGA shape" concept and its applications.

What is CGA shape?

CGA shape is a shape grammar for procedural modeling. This grammar permits to create buildings with high visual quality and geometric detail. It is based on rules applied to shapes to transform them in other shapes more precise and detailed. The power of CGA shape is that the creation of the hierarchical structure and the annotation of a model is specified in the modeling process. Another advantage of CGA shape is its efficiency. CGA Shape is the first concept to use a procedural approach to model detailed buildings with consistent mass models, and to show massive urban models with unprecedented level of geometric detail. CGA Shape is an extension of set grammars inspired by L-systems.

How does it work?

To create detailed buildings we first need to configure a finite set of basic shapes, and then to apply ordered rules on them in order to complexify the geometric structure.

In which order should rules be applied? (production process)

To apply a rule on a configuration you first need to select an active shape in the set, then to choose a production rule for this active shape in order to compute a successor, a new set of shape, next to mark the old shape as inactive and add the new one to the configuration.

What kind of rules can be applied?

The CGA grammar is composed of different rules, from basics to more complicated. The general rules used to modified shapes are translation, rotation according to an axis, and sizing. Then there is another basic rule: the split. The split allows us to split a shape along one axis in order to have several shapes. The scaling of rules may be absolute or relative in order to manage the adaptation of some shapes against other shapes. The repeat rule is used to repeat a shape along an axis. There is a particular split named the component split that permits to split into shapes of lesser dimension.

Example explained:

1- This first rule extracts a shape from the footprint (its height is fixed by the user), there is a translation in order to put the base of the building at the elevation 0. The last instance is the creation of a “hipped” roof on the top of the building. The roof command permits us to use later a variable that contains only the roof shape.

```
footprint -> S(1r,building_height,1r) facades
    T(0,building_height,0) Roof(‘‘hipped’’,roof_angle){ roof }
```

2- This rule transforms the side faces of the buildings in 2D surfaces stocked in the variable named façade.

```
facades -> Comp(‘‘sidefaces’’){ facade}
```

3- This rule identifies the street facing building side to place an entrance. “0.5” is the probability.

```
facade : Shape.visible(‘‘street’’)
-> Subdiv(‘‘X’’,1r,door_width*1.5){tiles|entrance} : 0.5
-> Subdiv(‘‘X’’,door_width*1.5,1r){entrance|tiles} : 0.5
```

4- This rule adds all the other faces of the facade to tiles variable.

```
facade -> tiles
```

5- Creation of the spaces for the windows on the facade.

```
tiles -> Repeat(‘‘X’’,window_spacing){tile}
```

6- Creation of the windows sides in the windows spaces.

```
tile -> Subdiv(‘‘X’’,1r,window_width,1r){wall|
Subdiv(‘‘Y’’,2r,window_depth,1r){wall|window|wall}|wall}
```

7- This rule permits to avoid the superposition of windows on other objects (occlusion).

```
window : Scope.occ(‘‘noparent’’) != ‘‘none’’ -> wall
```

8- Creation of the window shape and importing of win.obj in this space.

```
window -> S(1r,1r,window_depth) I(‘‘win.obj‘‘)
```

9- Creation of the spaces for the door on the entrance face.

```
  entrance -> Subdiv(‘‘X’’,1r,door_width,1r){wall |
  Subdiv(‘‘Y’’,door_height,1r){door | wall} | wall}
```

10- Creation of the door shape and import of door.obj in this space.

```
door -> S(1r,1r,door_depth) I(‘‘door.obj’’)
```

11- Import of wall.obj in the wall spaces.

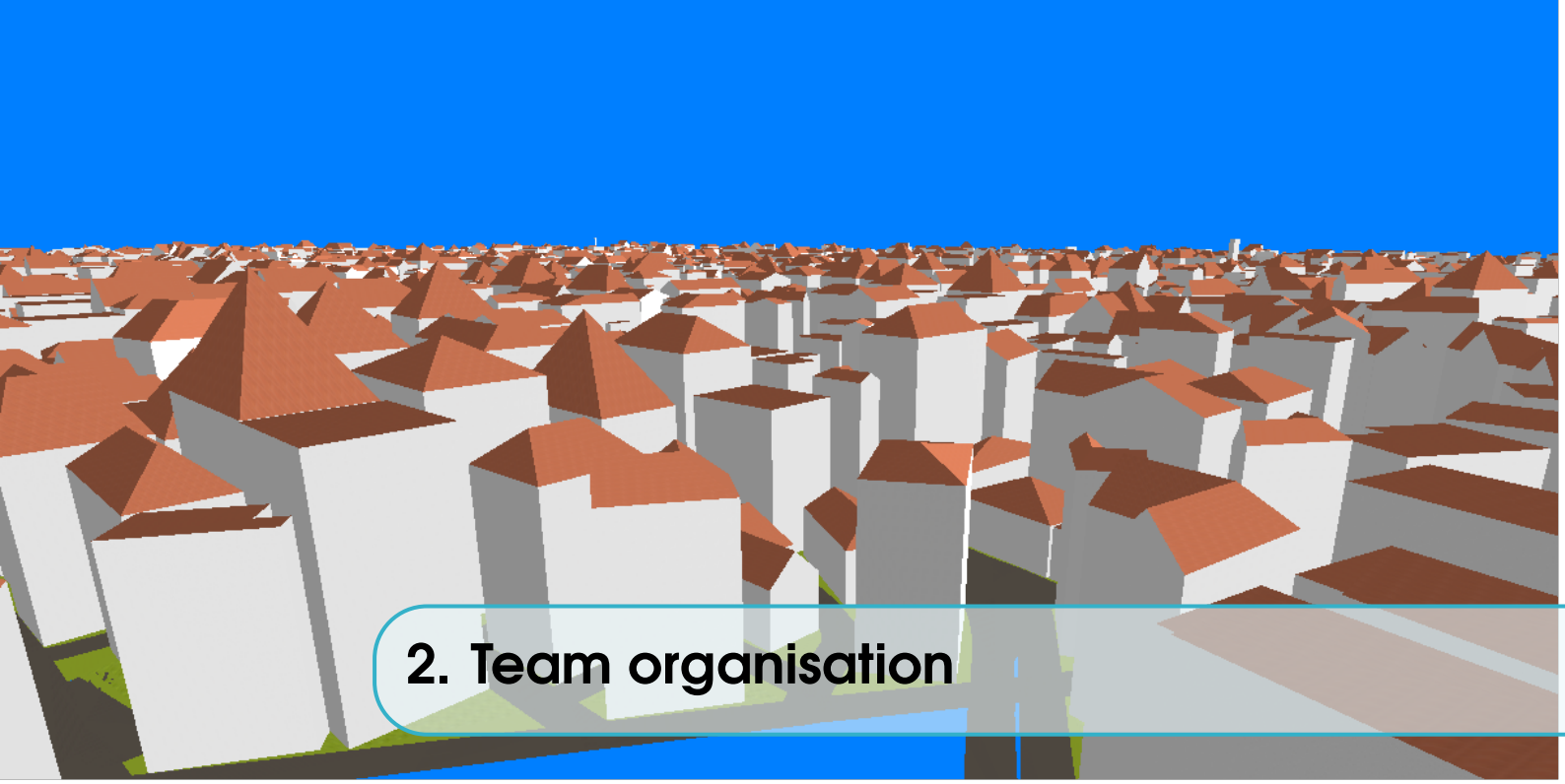
```
wall -> I(‘‘wall.obj’’)
```

12- This rule transforms the side faces of the roof in 2D surfaces stocked in the variable named covering and the edges of the roof in the variable named roofedges.

```
roof -> Comp(‘‘sidefaces’’){covering}
Comp(‘‘sideedges’’){roofedge} Comp(‘‘topedges’’){roofedge}
```

13,14,15,16- These rules are used to set the texture of the roof.

```
covering ->Repeat(‘‘XY’’,flatbrick_width,brick_length){flatbrick}
Subdiv(‘‘X’’,flatbrick_width,1r){epsilon|Repeat(‘‘X’’,flatbrick_width){roofedges}}
```



2. Team organisation

2.1 Job repartition

We worked on the global project with 4 other teams using the SCRUM method.

Our team is composed of four members:

- Amaury Zarzelli: Product Owner and developer.
- Ludivine Schlegel: Scrum Master and developer.
- Antoine Moutou: developer.
- Esseddik Ismael: developer.

2.2 Used tools

To apply the SCRUM method and carry out the project, we used different tools.

To organize our work we used:

- Easybacklog to create and manage our user stories, our product backlog and sprint.
- Trello to follow the development process in detail with assigned tasks, split the user stories to simple tasks and add more details on who we need to do.

The project uses C++ and deals with 3D objects. To manage that we used:

- Code::Blocks IDE with GNU GCC compiler to write an compile the project.
  - GEOS/GDAL libraries for GIS Features manipulation in C++.
  - Catch framework for unit tests.
  - OpenGL for the visualization.
- Doxygen to create automatically the developer documentation.
- Gitlab to manage, version and store our code online.
- Blender for an overview of our 3D «.obj» files created.
- QGIS for an overview and editing of the test input shapefiles.

We used the StarUML software to make the UML modeling, and the website Overleaf to write the report.

2.3 Sprint organization

The project lasted 4 weeks and was divided in four sprints.

2.3.1 First sprint

The first sprint consisted in the analysis of the articles, a clarification of the needs of the client, a research about the state of the art, the creation the UML and initialization of all the classes. We also started to read the shapefiles (SHP) using the GDAL/OGR library, created the roads and parcels from the SHP data and created the footprint (area on the parcel where the building can be build).

For this sprint we had underestimated the difficulty to use GDAL and the difficulty to create the footprint. First we had difficulties installing the GDAL/OGR, and some function calls were not intuitive.

example: calculation of the polygon area

→ in the documentation: polygon.get_Area()
→ in reality: polygon.OGRCurvePolygon::get_Area()

Footprint was a more difficult class to create than we expected because we had underestimated the complexity of the problem and did not think about all the limit cases.

2.3.2 Second sprint

During this sprint we finished the footprint creation, created the envelop (footprint with the height of the building without its roof). We also developed a triangular model and started to implement the 3D objects. The difficulty of this sprint was the 3D objects implementation inspired by the CGA Shape paper.

2.3.3 Third sprint

During this Sprint we created the different types of the parcels (Industry, Office, ApartmentBuilding, Townhouse, Villa) and the real building with its roof, we converted the shp data and our 3D models of Envelop and Buildings to Wavefront *.obj* format. The creation of the buildings on the footprints and the *.obj* export were the difficulty of this sprint.

2.3.4 Last sprint

This Sprint was dedicated to writing of report and user documentation and the rendering with OpenGL of the textured 3D objects (Buildings, Parcels and Roads). The implementation of the OpenGL overview was the difficult part of this sprint. We adapted the ING2 OpenGL project of a group member[1], and we needed to include other libraries and adapt the scripts for it to work on our project and data. The last step of this sprint will be the integration of our project with the master branch.

2.4 Work done

Amaury Zarzelli

As a Product Owner, I contacted the client when the team had questions. I also worked along with the product owner of the subject 4 team to make sure their output files corresponded to our expected inputs. Lastly, I updated the product backlog along with Ludivine.

As a developer, I helped in the making of the UML diagram and the definition of the classes (I was the one to suggest to decompose our 3D objects into triangles). I developed the function that creates a Footprint from a parcel object (Parcel::create_footprint()). I also created the function that determines automatically a parcel's type according to its distance form the center of the city (called in the constructor). I tuned the building types parameters. I developed the functions to convert 3D objects (and lists of 3D objects) into .obj files. I wrote the .mtl files. I adapted my OpenGL project from last year to the subject for the visualization. I also helped Ismaël in the standardization of the developer documentation.

[1] https://github.com/azarz/EAGL

Antoine Moutou

I deeply analysed and sumed up the article Procedural Modeling of Buildings (2006).I helped in the definition of the classes and the making of the UML diagram. I developed the Point, Triangle, BuildingModel methods using 3D geometry. I took a long time to make the split function. I create a part of the unit test and I gathered all the tests in one single file in order to be executed out of the project.

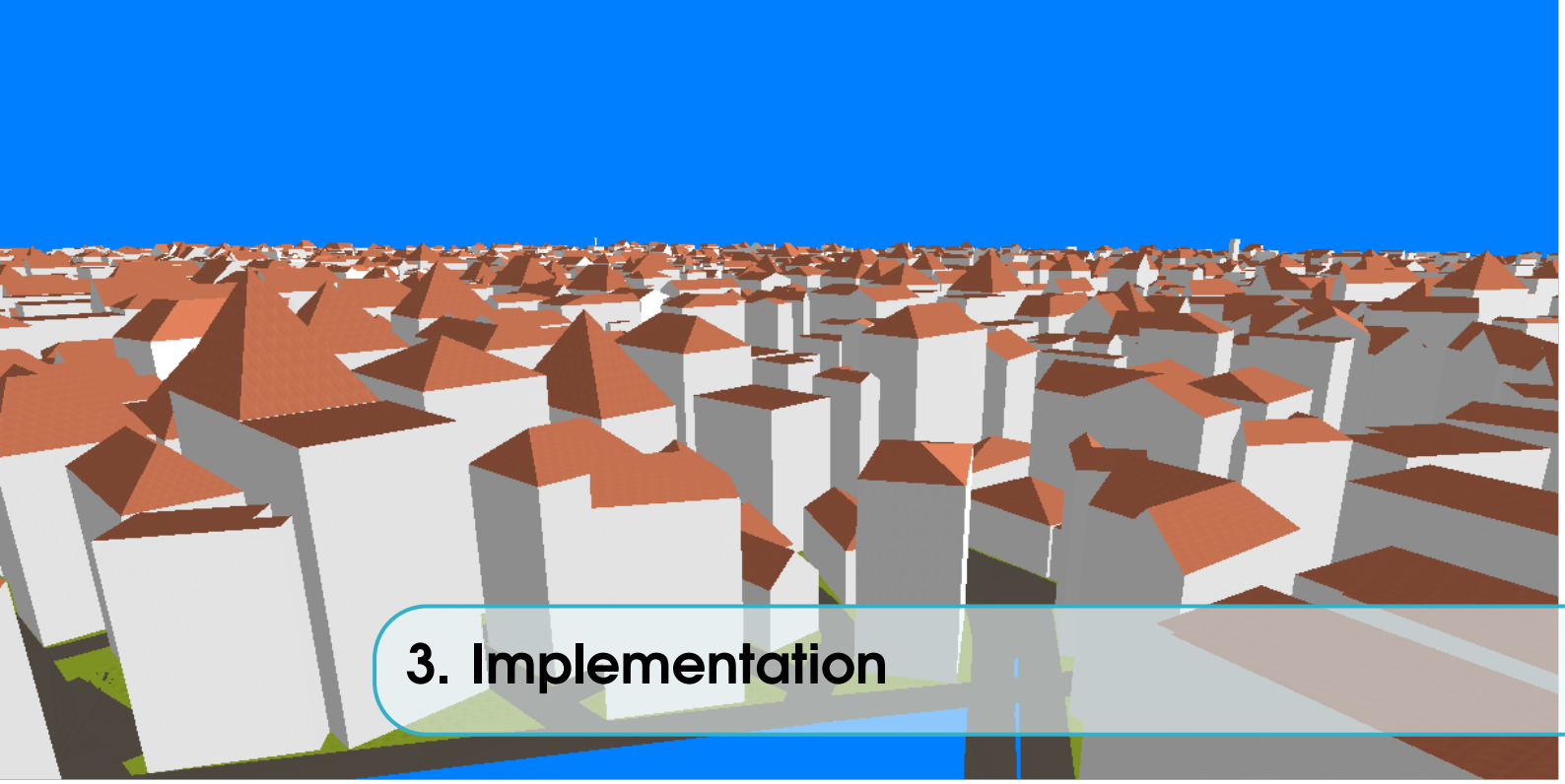
Ismaël Esseddik

As Developer, I worked on the creation of the building roof, it took me the most of my time. I worked on the creation of the envelop class and implementation of the calculation for the number of floors and the lot price. I also standardized a big part of the documentation inside the header files(finalized by Amaury). All these tasks constitute the essential of my work next to which I did some research for existing tool that could be useful or helpful.

Ludivine Schlegel

As the Scrum Master I manage the Backlog and the Sprint, I update EasyBacklog & trello and I take part in the meetings with other Scrum Masters.

As a developer I make ULM with starUML, I create all the class file and this argument, I implement the Point's constructor, the Triangles' constructor, the constructor of BuildingType and its 5 real building type class , the function BuildingType.get_height(), the function BuildingType.get_margin(), the Road's constructor, the beginning of Parcel's constructor, the beginning of function Envelop.to_obj() (change OGRlinear ring to OGRpolygon then Triangle list), the Building's constructor and the function building.to_obj(). I also implement ABuGeR() function for create fill and put the old main on it and create a new main using this function and the function creat_wall(), gauss3P(), gauss2P(), larger_rectangle_included(), area(), areaL(), sum(), all_ones(), L_ones(), rotate_p(), larger_L_include(), angle_calcul(), open_shp_road(), open_shp_parcels(), poly_to_triangle(), flat_roof() and I adapt linear_spine() to L building;.



3. Implementation

3.1 Detail and explanation of the most difficult implementations

3.1.1 Building creation

The goal of this step was to create a building into the footprint space. The difficulty was to get the best shape on the footprint. So, we started with a rectangle: we looked for the larger rectangle include on the shape. This method was not implemented in GDAL and we did not find an existing C++ library to do so. We found an algorithm who finds the best rectangle on a 1/0 matrix so we used it. We created a grid of point and the matrix associated for each envelop footprint: 1 if the point is on the shape, 0 if it is out. Then we looked for the biggest rectangle on this matrix. We have the largest rectangle for one orientation so we did it again with different orientations, first the orientation is fixed then we change that to take the main angle of the shape. After that we created the L building: it was more complicated since the L has 4 possible orientations and more possible shapes. We created a first method to find two "1" rectangles: for a given sub-matrix it searches the first possible L then it shrinks the rectangle one step by one to fill it with "1". But this method was too slow, so we did a second method who adjusts the limit according to the 0 found, and it was much faster.

3.1.2 Roof creation

Figure 3.1: roof types

The goal of this implementation was to integrate a triangular mesh into a building, representing its roof. That implied to consider for each type of basic roof shape (flatten, gabled, hipped) the number of vertices in the roof base polygon. In these base polygons we managed to get only a defined number of vertices (4 for squares & rectangles or 6 for L patterns building). Applying the Thales' theorem using the centroid polygon, we made functions which takes as arguments an angle for the height of the spine and the building on which it will be added. After getting the polygon and the spine's vertices on a list, we set up according to the length and the width of the polygon the triangular mesh of the roof just before adding it as a new building model.

3.1.3 Split method

The aim of this method was to split some triangular faces, selected by their type, according to an axis and an origin. First we chose to solve this problem with matrix calculation using the pivoting method of Gauss. But this method didn't pass the unit test because there was a problem with double numbers. The multiplication and division of a double number, sometimes, changes its exact value. This problem had as consequence to avoid intersection between the projected straight line, created with the axis and the point, and the sides of the triangle. In order to correct this error we chose to work in another reference system defined by two sides of the triangle to work in 2 dimensions. This technique allows us to have an intersection and to solve it in 2 dimensions. It was also easier to project the straight line on the plan of the triangle. Once the intersections found, we just need to check if they are on the sides of the triangle or out. Depending on the number of real intersections we split our triangle in two or three or we don't split the triangle if there is no intersection. This second approach works better because it passed all the unit test.

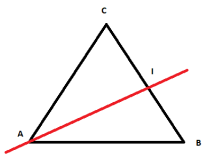


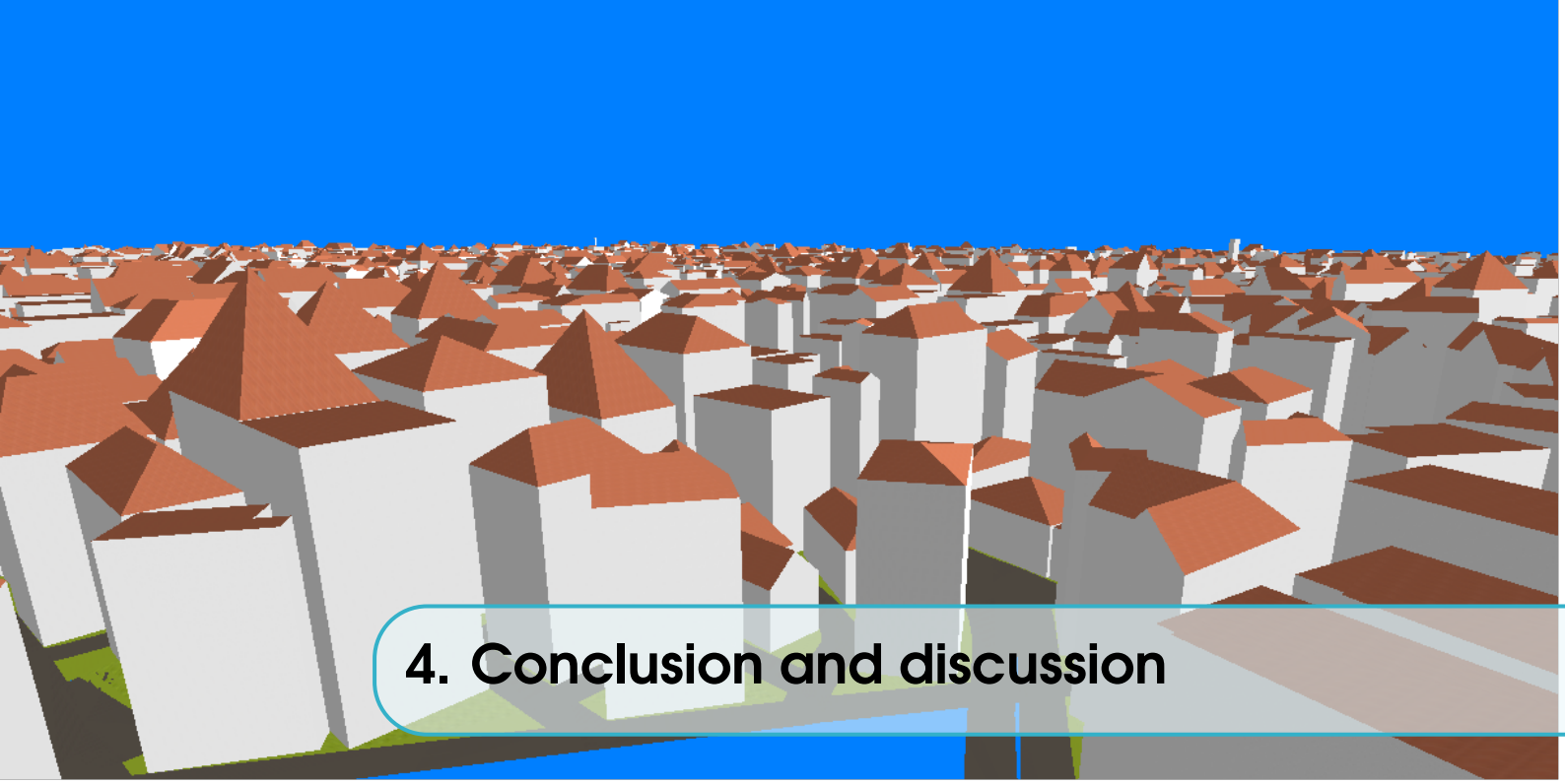
Figure 3.2: Split passing by a vertex

3.2 Conversion in .obj format and visualization using OpenGL

The Wavefront .obj format is a simple 3D format in which the elements are stored line by line. For example, a vertex located at the coordinates (0,0,0) will be written as "v 0 0 0". The faces are defined using the vertices indices in the file. A face made of the 1st, 2nd and 3rd vertices will be written as "f 1 2 3". Furthermore, texture coordinates and vertices normal can be defined, in a similar way as the vertices and faces. We developed for each of our 3D objects a method to convert an object (and then a list of objects) into a string corresponding to the .obj format. The most tricky part was the calculation of the vector normals (corresponding to the orientation of the plane), that are not necessary for Blender but that are needed in the OpenGL implementation we used. Each .obj files comes with a .mtl file that defines the aspect of the object (lightning and texture).

3.3 Unit tests creation

We used the library Catch to perform our unit test. All the tests are in the catch.cpp file. We tested almost all our functions in order to check after any modification if all the other functions work well.



4. Conclusion and discussion

4.1 Discussion

4.1.1 Unused functions

Our primary goal was to obtain detailed buildings with methods that implement CGA Shape. But we didn't get the time to do that. So there remains some functions that are not used. All the methods to size, to translate, to rotate, to join, to split have passed the unit tests successfully. So it would be interesting to use them if our work is continued by an other team.

4.1.2 Unimplemented shapes

Regarding the roof, some of basic roof shape like the gambrel, cone and mansard type aren't yet implemented (two first and last ones from Figure 4.1). T U and H building pattern also have not been implemented.

4.1.3 Improvement of the roofs

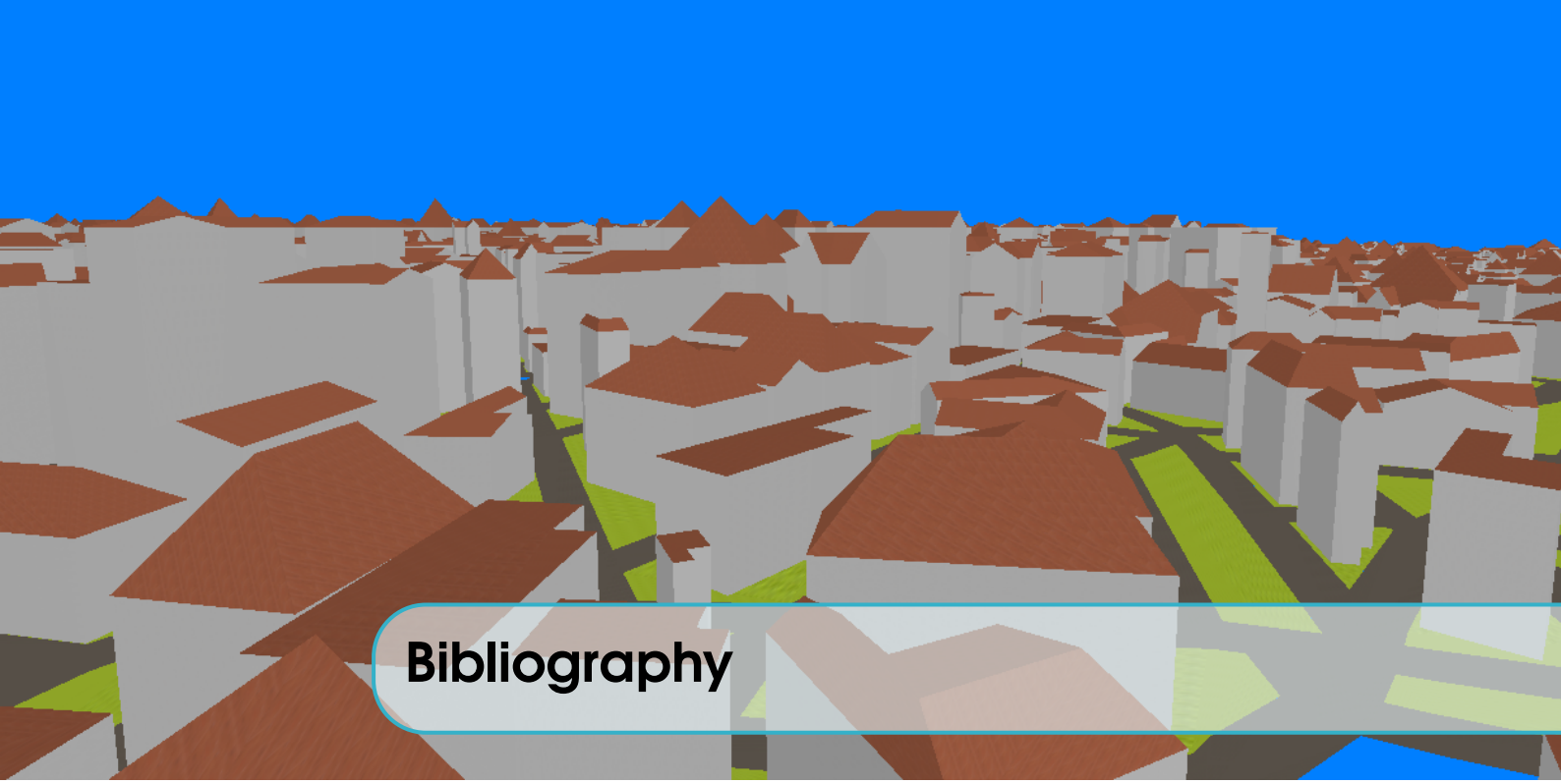
For the roofs, it should be possible to compute more complex forms with the straight skeleton algorithm. It can be implemented using the CGAL library, but we did not have enough time to try it.

4.1.4 Improvement of the textures

Currently, the textures are square shapes repeated over the whole surfaces. A way to improve the visual aspect of the city would be to implement textures of windows on the building walls. This would imply a more complex calculation of the texture coordinates in the .obj files, taking into account each vertex position, and each building height.

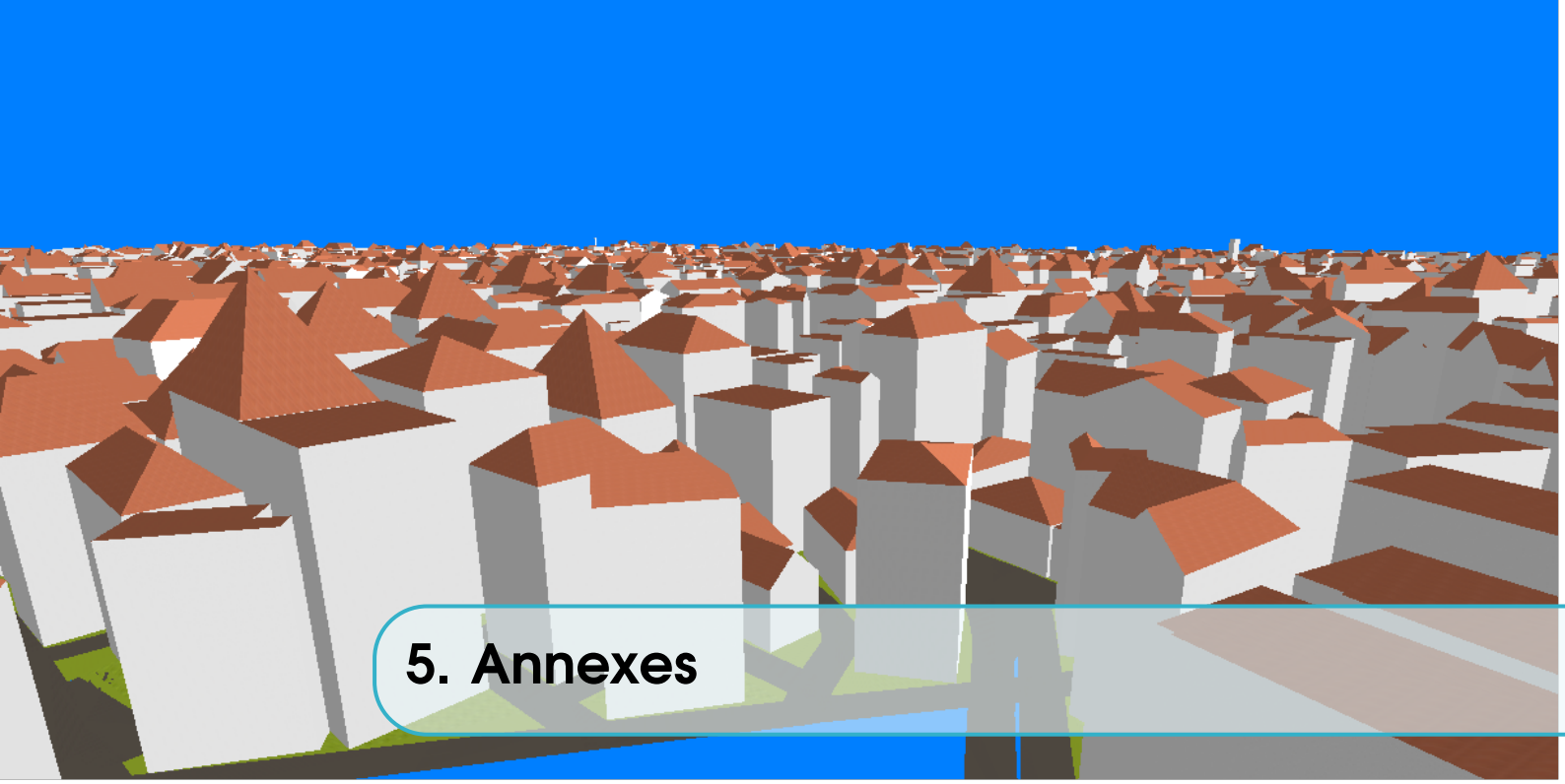
4.2 Conclusion

Our goal was the 3D modeling of buildings. Given our abilities and the deadline to carry out this project we did our best to get organized in the most efficient way possible, based on the Scrum methodology. After a month of work, we finally succeeded in generating a large amount of textured building envelop in linear and L pattern, with 3 or more different roof types and displaying them using OpenGL.



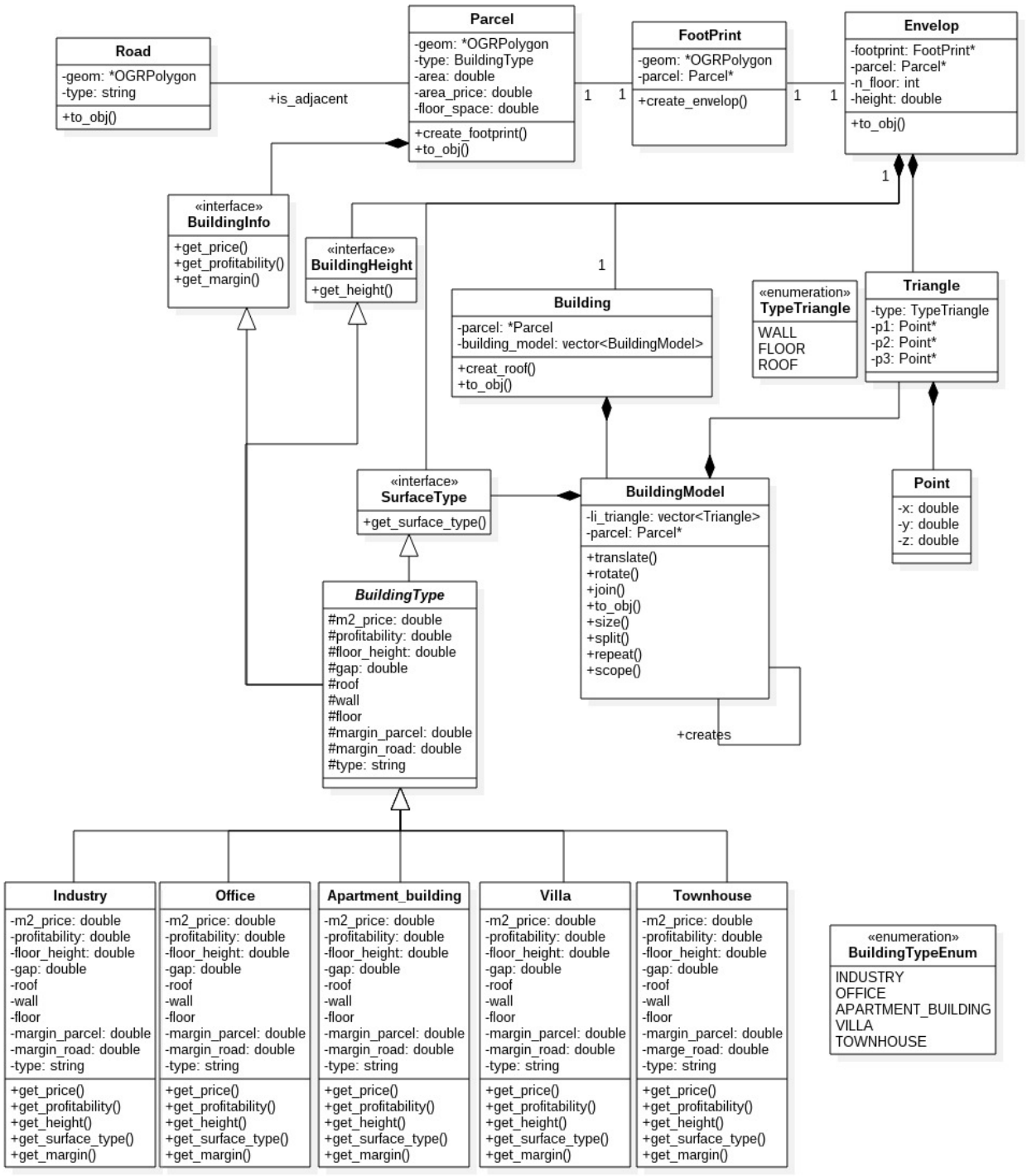
Bibliography

- *Basil Weber, Pascal Müller, Peter Wonka, and Markus Gross*, **Interactive Geometric Simulation of 4D Cities**, Wiley-Blackwell, Computer Graphics Forum, March 2009, vol.28(2),11p.(481-49), Oxford, England. `http://peterwonka.net/Publications/pdfs/2009.EG.Weber.UrbanSimulation.FinalVersion.pdf`
- *Pascal Muller, Peter Wonka, Simon Haegler, Andreas Ulme, Luc Van Gool*, **Procedural Modeling of Buildings**, ACM, Transactions on Graphics, July 2006, Vol.25(3), 10p.(614–623), New York, USA. `http://peterwonka.net/Publications/pdfs/2006.SG.Mueller.ProceduralModelingOfBuildingfinal.pdf`



5. Annexes

5.1 UML diagram



5.2 UML class

Point:
Contain the (x, y, z) coordinates of one point
function: print(): print the x, y, z coordinates of the point; translate(): translate the point according to an array; rotate(): rotate the point with a rotate axis and an angle; size(): resize the point according to an array;

Triangle:
Contain a 3D triangle composed to 3 Point and the type of triangle (floor, wall, roof)
function: print(): print the coordinates of the 3 points of triangle; set_type(): change the type of the triangle; translate(): translate the triangle according to an array; rotate(): rotate the triangle with a rotate axis and an angle; size(): resize the triangle according to an array; split(): split the triangle according to an axis and an origin; repeat(): make a copy of the triangle;

BuildingType:
Contain the data for the each type of building:
*argument:*Profitability (price of a m^2 of floor)(€/m^2); floor height (m); gap (height of the first floor)(m); margin parcel (distance between the building and another parcel) (m); margin road (distance between the building and a road) (m); type (type of the building).
function: get_height(number_of_floor): give the height of building depending the number of floor get_margin(): give the margin between the building and a no road border and the margin between the building and a road border.

Building type	profitability	floor height	gap	margin parcel	margin road	type
Office	4.0	3.0	5.0	0.001	0.001	Office
Industry	1.0	3.0	15.0	2.0	10.0	Industry
Apartment building	3.0	3.0	5.0	0.001	4.0	ApartmentBuilding
townhouse	1.0	3.0	3.0	0.1	0.5	Townhouse
Villa	1.0	3.0	3.0	5.0	9.0	Villa

Table 5.1: Building rule parameters according to the type

Road:
Contain the polygon of road and this importance (1 for main road and 2 for the secondary road)
function: to_obj(): export the geometry of building into *.obj* text;

Parcel:
Contain one parcel:
argument: geometry (polygon of parcel); Building type (BuildingType corresponding of the parcel Building); area (area of the parcel) (m^2); parcel price (€);floorspace (area the building need to be profitable)(m^2);
function: print(): print the parcel information; create_footprint(): create the foot print corresponding to the parcel; compute_type(); create the type of building build on the parcel according to the length between the parcel and the center of the city; to_obj(): export the geometry of building into *.obj* text;

Footprint:
Contain the Building space.
function: create_envelop(): create the envelop corresponding;

Envelop:
Contain the Building space and the building's height without roof.
function: to_obj(): export the geometry of building into *.obj* text;

Building:
Contain the Building with it roof stored as a triangles in BuildingModel.

function: create_roof(): add the roof on the building; to_obj(): export the geometry of building into *.obj* text;

BuildingModel: Contain a list of triangle
function: translate(): translate the selected faces of the BuildingModel according to an array; rotate(): rotate the selected faces of the BuildingModel with a rotate axis and an angle; size(): resize the selected faces of the BuildingModel according to an array; split(): split the selected faces of the BuildingModel according to an axis and an origin; repeat(): make a copy of the BuildingModel; join(): join a BuildingModel with an other;

Additional methods

External methods.

ABuGeR():

Transform road and parcel shapefiles to .obj and creates building .obj.

create_env_roof():

Creates flatten roof for a Envelop.
Used in: Envelop.to_obj();

create_wall():

Creates walls from the building footprint and envelop.
Used in: Envelop.to_obj() & Building creator;

gauss3P():

Finds (if it exists) the solution of AC=B with 3 parameters
Used in: old split method;

gauss2P():

Finds (if it exists) the solution of AC=B with 3 parameters.
Used in: old split method;

larger_rectangle_included():

Find the larger 1 rectangle on a matrix composed of 1 and 0. Used in: Building creator;

area():

Calculates the area of the rectangle Used in: larger_rectangle_included;

areaL():

Calculates the area of an L shape on submatrix.
Used in: larger_L_included;

sum():

Calculates the sum of 1 on submatrix.
Used in: larger_L_included;

all_ones():

Test if the submatrix is composed only of 1.
Used in: larger_rectangle_included;

L_ones():

Test if the submatrix have a L shape composed only of 1.
Used in: larger_L_included;

rotate_p():

Rotation of a point around a point
Used in: Building creator;

larger_L_included():

Find the larger L shape on a matrix composed of 1 and 0.
Used in: Building creator;

angle_calcul():

Calculate the angle of a point with the X axis.
Used in: Building creator;

invert_matrix():

Invert a 3x3 matrix.
Used in: split method;

matrix_product():

Calculate the product of a 3x3 matrix with a 3x1 matrix.
Used in: split method;

matrix_translation():

Calculate the translation of a matrix 3x1 by a vector 3x1.

Used in: split method;

open_shp_roads():
Opens the SHP of roads and creates the road object associated, along with calculating the centroid of the roads.
Used in: ABuGeR();

open_shp_parcels():
Opens the SHP of parcels and create the Parcel object associated.
Use on : ABuGeR();

get_intersection_road():
Gets the intersection line between a linear ring and the road polygons.
Use on : ABuGeR();

get_other_sides():
Get the other sides of the parcel bounds (parcel bounds with not touch road).
Use on : ABuGeR();

poly_to_triangle():
Creates the triangulated surface of a polygon.
Use on : ABuGeR();

crossed_spine():
Manage a crossed roof skeleton, for rectangle only.
Use on : Building.creat_roof();

linear_spine():
Manage mono linear roof skeleton.
Use on : Building.creat_roof();

linear_cross_spine():
Manage linear roof skeleton linked to the four corners roof.
Use on : Building.creat_roof();

flat_roof():
Manage mono linear roof skeleton.
Use on : Building.creat_roof();

triangles_to_obj()
Returns 3 strings corresponding respectively to the vertices, the uv_coordinates and the faces of a vector of Triangle in Wavefront .obj format.
Use on : Road.to_obj(), Parcel.to_obj(), Envelop.to_obj(), Building.to_obj();

Product Backlog

Geomatic Project 5 Velocity: 3.0

Theme	ID	User Story	Acceptance Criteria	Comments	Sprint	Status	Poi	Days
Initialization	INI1	As a team member I want to have a defined role So I can know what I have to do	a) Everyone in the team has a defined role		Sprint 1	Accepted	0.0	0.0
	INI2	As a team I want to have a common workspace - repository So I can version, share and	a) Online repository created		Sprint 1	Accepted	0.0	0.0
	INI3	As a team I want to have tools to plan the tasks	a) Online product backlog b) Online dashboard		Sprint 1	Accepted	0.0	0.0
	Total for theme 'Initialization'						**0.0 points / 0.0 days**	
Analysis	ANA1	As a team member I want to have a summary of each article	a) Summaries written		Sprint 1	Accepted	3.0	1.0
	ANA2	As the product owner I want to clarify the needs of the client	a) Needs clearly defined		Sprint 1	Accepted	2.0	0.7
	ANA4	As the product owner I want to study the state of the art So I can know what's already	a) Report on the state of the art		Sprint 1	Accepted	2.0	0.7
	ANA3	As the product owner I want to contact people that did similar projects So I can have another perspective on what to do	a) Sylvain's explanations on his modelling		Sprint 1	Accepted	1.0	0.3

Figure 5.1: Page 1 of the product backlog

	Total for theme 'Analysis'					**8.0 points / 2.7 days**	
Conception	CON2	As a developper I want to know what are the inputs and outputs So I can adapt the processings	a) Clearly defined input format b) Clearly defined output format	Sprint 1	Accepted	5.0	1.7
	CON1	As a developper I want to have a precise modelization of the application (UML)	a) UML diagrams written	Sprint 1	Accepted	20.0	6.7
	Total for theme 'Conception'					**25.0 points / 8.3 days**	
Implementation	IMP1	As a developper I want to place the building on the parcel So I can fill it	a) Buildings footprint automatically calculated for each	Depends: CON2	Sprint 2	Accepted	13.0	4.3
	IMP2	As a developper I want to claculate the spatial envelope of a building	a) Envelope successfully calculated	Depends: IMP1	Sprint 2	Accepted	5.0	1.7
	IMP3	As a developper I want to implement the CGA Shape paper	a) Building geometry sucessfully calculated (using CGA shape)	Depends: IMP2, CON2	Sprint 2	Accepted	40.0	13.3
	IMP8	As developper I want to determine the parcel type	a) Each parcel has a type		Sprint 3	Accepted	13.0	4.3
	IMP4	As a developper I want to convert shapefiles to 3d models	a) Successful conversion b) Files readable by		Sprint 3	Accepted	13.0	4.3

Figure 5.2: Page 2 of the product backlog

IMP5	As a developper I want to convert the buildings CGA shapes to 3d models So I can render them	a) Successful conversion b) Files readable by Blender	Depends: IMP3	Sprint 3	Accepted	13.0	4.3
IMP6	As a developper I want to render the city using OpenGL	a) Visuatization of the city in a dedicated window		Sprint 4	Completed	13.0	4.3
IMP7	As a user I want to see beautiful buildings	a) Texturing of the buildings		Sprint 4	Completed	20.0	6.7
Total for theme 'Implementation'						**130.0 points / 43.3 days**	
Validation							
Total for theme 'Validation'						**0.0 points / 0.0 days**	
Delivery							
DEL1	As a client I want to know what the team has done during the project So I can assert its work	a) Report written b) Good presentation		Sprint 4	In progress	8.0	2.7
DEL2	As a client I want to understand the code So I can use it later	a) Developer documentation		Sprint 4	In progress	3.0	1.0
DEL3	As a client I want to know how to use the program	a) User documentation		Sprint 4	Completed	2.0	0.7
Total for theme 'Delivery'						**13.0 points / 4.3 days**	
Total for backlog 'Geomatic Project 5'						**176.0 points / 58.7 days**	

Figure 5.3: Page 3 of the product backlog