TDK dolgozat

Polcz Péter

Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar
mérnök-informatika

3D virtuális városrekonstrukció LiDAR pontfelhőkből

3D virtual city reconstruction from LiDAR point clouds

Polcz Péter
IV. évfolyamos hallgató

témavezető:
Dr. Benedek Csaba
MTA SZTAKI EEE laboratórium, tudományos főmunkatárs

2014. 04. 15.

Kivonat

A háromdimenziós virtuális városmodellek széles körben elterjedtek az utóbbi években, szerepük a megjelenítés mellet különböző elemzési feladatokban is jelentős. Ugyanakkor az adatok mennyisége miatt a modellgenerálás nagyfokú automatizálása kulcsfontosságú a hatékonyság szempontjából.

A LiDAR (Light Detection and Ranging) mérési technológia segítségével nagy pontosságú levegőből készített háromdimenziós pontfelhőket kaphatunk különböző földrajzi területekről, ami az optikai úton sztereokép-számítással nyerhető magasságtérkép hatékony alternatívájának számít a városrekonstrukciós feladatokban.

Tudományos diákköri munkám során olyan automatikus és robosztus algoritmusok kifejlesztésére törekedtem, melyek légi LiDAR felvételek feldolgozásával részletekben gazdag háromdimenziós virtuális városmodelleket generálnak. Fontos célnak tűztem ki, hogy a rendszer képes legyen különböző építészeti irányzatot követő háztetőkről is valósághű modelleket rekonstruálni, másrészt akár több négyzetkilométernyi területet lefedő LiDAR mérések feldolgozását is felhasználói beavatkozás nélkül tudjuk elvégezni.

A feldolgozás bemenetét szolgáló Budapest belvárosáról készült nagypontosságú légi LiDAR felvételeket az Infoterra Astrium GEO-Information Services Hungary Kft. bocsátotta rendelkezésünkre. A felvételek mintegy 300 m-es repülési magasságból készültek, az adathalmaz az egyes repülési irányoknak megfelelően 3-6 km hosszú és kb 200 m széles szegmensekből áll, ahol az egyes szegmensek 10-20 millió pontot tartalmaznak.

A feladat megvalósításakor felhasználtam a kutatócsoportunkban korábban kifejlesztett pontfelhőosztályozó és előzetes tetőszegmentáló eljárásokat. A munkafolyamathoz a következő algoritmusok kidolgozásával és implementációjával járultam hozzá. A felületnormális alapú tetőszegmentálás alapötletéből kiindulva terveztem egy olyan eljárást, amely a tetőszegmensek határait is figyelembe veszi, így a szomszédos szegmensek résmentesen összeérnek. Algoritmust dolgoztam ki a tetőrészek éleinek detektálására, és ezek alapján történő poligon felületek generálásra. Gondot jelentett, hogy mivel a felhasznált háromszögelési eljárások a tetőszegmensek konvex burkára illesztik a felületmodellt, konkáv tetőszegmensek homorú külső részeiben hamis háromszögek keletkezhetnek. Erre a problémára egy Markov véletlen mezős eljárást adtam, melynek segítségével kiszűrtem a hibásan megjelenő háromszögeket. Az eljárásokat megvalósító szoftvert alkalmassá tettem nagy területet lefedő mérési adatok feldogozására, módszereim hatákonyságát egy $240m \times 2.3km$ területű és kisebb $200m^2$-es régiók mérési adatain teszteltem. A megfigyelt eredmények alapján a szoftver robosztusnak bizonyult.

Az elkészült algoritmusokat az MTA SZTAKI Elosztott Események Elemzése Laboratóriumában folyó és az Európai Űrügynökség által támogatott DUSIREF projektben szeretnénk felhasználni a jövőben, különböző légi és földi pontfelhő alapú gépi felismerő eljárásaink eredményeinek a megjelenítéséhez. Másrészt az elkészült háromdimenziós modelleket különböző időpontokban készül űrképekkel szeretnénk összevetni, megvizsgálva az adaptív textúrázhatóság és a változásdetekció lehetőségeit.

Abstract

Three dimensional urban scene modelling became important issue in the last few years. Beside visual experience, 3D city modelling has gained a significant function in diverse analysing tasks, however the amount of data requests a high level of automation of model generation.

LiDAR (Light Detection and Ranging) measurement technologies provide high resolution three-dimensional aerial point clouds with a high accuracy from different geographical areas. This type of data is considered as an effective alternative solution of elevation models generated by optical stereo images.

In my work for the Students' Scientific Competition, I aim to design automatic and robust algorithms in order to produce detailed 3D virtual city models by analysing the aerial LiDAR measurements. An important goal to work out is to construct realistic models for rooftops following different architectural trends, and to process large LiDAR measurements covering areas of several square kilometers without user interraction.

As input, we have used high resolution LiDAR records of Budapest city center, which have been provided by Infoterra Astrium GEO-Information Services Hungary. The measurements were taken in an altitude of $300m$, and the data set corresponding to each flight directions covers a $3-6\ km$ long and $200m$ wide area, where each set contains 10-20 million points.

Pursuing my task, I have used a point cloud classification and a preliminary roof segmentation procedure developed by our research group. I have contributed to the process workflow with the following algorithms and implementations. Proceeding from the idea of the surface normal based roof segmentation I have designed a procedure, which takes into account the boundaries of each roof segment, so that the adjacent segments connect without gaps. I have developed an algorithm to detect 3D edge lines of the rooftops, and a further polygon generator on the basis of these boundary lines. Since the applied triangulation methods operate on the whole convex hull of the input points, hollow outer parts of the roof segments are filled in with false triangles. To solve this problem, I have proposed a method using a Markov Random Field, in which I filter out the incorrect triangles lying on the concave parts. In order to process large datasets, I have developed a software on the basis of the previously mentioned functions, and I have been tested it on a cloud of 8 million points covering a $240m \times 2.3km$ large region and on other datasets of smaller territories having area up to $200m^2$. Based on the observed results the software proved to be robust.

In the future, the designed algorithms are intended to be used in the ongoing DUSIREF project of Distributed Events Analysis Research Laboratory of MTA SZTAKI which project is funded by the European Space Agency. Here the synthesized city model will provide a 3D visualization environment, where the results of different terrestrial or aerial LiDAR based computer vision and machine recognition procedures can be displayed. Furthermore, the obtained three-dimensional models will be compared with satellite photos taken at different times, analysing the possibilities of adaptive texturing and change detection.

1. Introduction

In the last decade, LiDAR (Light Detection and Ranging) has been widely used in various remote sensing application fields. LiDAR is an optical remote sensing technology that can measure the distance of targets from the scanner by illuminating the target with laser light and analysing the backscattered light, therefore a such a laser scanner yields a 3D point cloud representing the objects around the scanner.

A specific type of these sensors can be mounted on airplanes, and the provided scans are appropriate for creating digital terrain models (DTM) and digital elevation models (DEM). These models are efficient and detailed descriptions of fields, valleys, mountains or other desert areas. These irregular, rough and mountainous terrain types cannot be represented as a set of regular shapes.

On the other hand, in case of cities or other urban settlements polygon reconstruction constitutes another alternative solution for modelling. The main targets of the reconstruction are the buildings having regular geometrical shapes introducing the possibility to approximate them with several three-dimensional polygons. Worldwide projects (Google maps 3D, Nokia maps) are devoted to this topic.

As input, we have used high resolution LiDAR records of Budapest city center, which have been provided by Infoterra Astrium GEO-Information Services Hungary.

It is a fact that, monumental civil apartment houses with circular corridors built in the XX. century are very frequent constructions in Budapest, especially in the VI., VII., VIII. districts. However these building blocks are very complex architectural beings introducing further challenges.

In this documentation I intend to present my approaches of aerial point cloud processing, three-dimensional city reconstruction and urban scene modelling.

1.1 Aerial LiDAR

Aerial LiDAR employs a laser fired to the ground from a GPS-monitored aerial sensor (fastened on the bottom of an aircraft) to accurately measure the distance of the airplane from the ground surface points, providing elevation data. GPS sensors are installed both on the aircraft, and at a known ground location (base station), which record frequently the position of the aircraft relatively to the GPS satellites. Simultaneously, an aerial IMU sensor records the pitch/yaw/roll of the aircraft, ensuring that the measured LiDAR returns can be converted into precise 3D point cloud datasets.

First and last return

By reflecting from objects such as tree canopy, structures and complex surfaces, a portion of the LiDAR pulse returns back to the sensor, where these returns are registered. Another part of the laser beam propagates towards and reflects from an object at a lower altitude. The first return reflects higher elevation objects or surfaces such as tree canopy, high voltage conductors between buildings, curled faces, irregularities etc. The last LiDAR return reflects the ground level and other regular smooth faces like vehicles and rooftops.

1.2 Related works on urban scene modelling

This research domain has considerably progressed during the last decade. Many of computer vision researchers have developed new techniques and algorithms in order to create not only realistic but also simple[1] city-models at the same time. Regarding the latest publications, significant results have been encountered by Lafarge et al. [4, 5], Zhou et al. [9–13], Huang et al. [7, 8] and Verma et al. [14]. Zhou's approach consists in geometrical and topological corrections of an initial mesh on the basis of local observations of the buildings' orientation. Whereas Lafarge and Huang defined geometric 3D primitives to fit them to the different building types and rooftop shapes appearing in the point cloud. Lafarge et al. [4, 5] also handled non-planar primitives as cylinders, spheres and cones.

A new technique has been proposed for shape recognition and template fitting, commencing with edge detection, which is realized by minimizing a quadratic distance error of the LiDAR points from the corresponding fitted 3D segments. Secondly, the planar structures, including the most common roof shapes, were extracted by applying a region growing (i.e. flood-fill) process. Thirdly, non-planar shapes were detected from the remaining points, which had not been fitted by any of the planes. The detected roof structures were marked with different labels, so that the points belonging to a specific structure were assigned the same label. In the next step, the 3D points were projected onto a horizontal 2D grid[2] storing each point's label and its z coordinate value. The second parameter (the elevation value) was essential because without it the inverse operation of projection (elevation into the 3D space) cannot be done. Afterwards a label propagation had been adopted on the grid using a Markov Random Field, which provided an arrangement of structuring elements (planes, cylinder, etc...). After this arrangement Lafarge proposed an inverse operation (elevation into 3D space) resulting in an impressive regular 3D model. Finally, virtual 3D models were obtained by elevating the structuring elements.

Huang et al. [7, 8] created complete roof models (composed by planar primitives) and attempted to fit them to the cloud regions classified as buildings using different statistical methods (in particular likelihood function maximization). After a geometrical adjustment, the primitives were "merged" into a plausible model.

Verma et al. [14] also used a statistical approach, by building a dual graph from the roof segments. However, this technique only worked for planar roof models.

[1] in the means of reduced number of facets

[2] the 2D grid is represented by an image

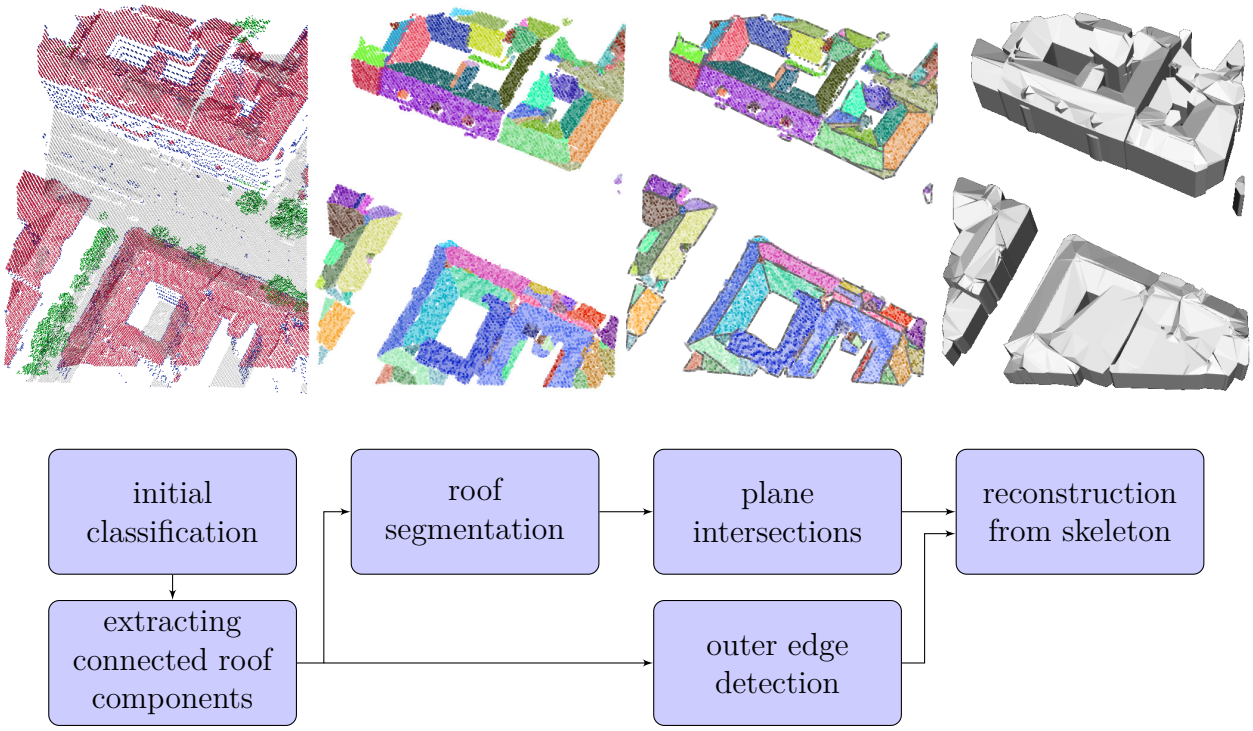


Figure 1.1. Workflow of the proposed method

1.3 A brief description of my proposed reconstruction algorithm

The workflow includes four steps as illustrated in Figure 1.1. First, the point cloud is classified using an unsupervised method (presented in Chapter 3), distinguishing four different classes: ground, building, vegetation and clutter.

In Section 3.1, the point cloud regions classified as buildings are divided into several parts in order to reduce the complexity of the further steps. Each part of the cloud will contain a reduced number of points belonging to a single complex rooftop.

The proposed algorithm approximates each rooftop by planar shaped faces. These planar roof segments are extracted by a robust method detailed in Chapter 4. The points of a roof component determine a plane, which is calculated through minimizing the sum of squared distances of the points from the plane.

The next step consists in generating a 3D skeleton model for each building by detecting the roof's so called *feature lines*. Before exactly defining the *feature lines* let us introduce the concept of *inner* and *outer edges*. *Inner edges* lie alongside the connection of two neighboring faces of the roof, and they are determined by intersection of the neighboring planar roof components. *Outer edges* are intersection lines of the vertical walls and the roofs. We note here that in the airborne LiDAR point clouds, we have usually no reflection from the vertical walls, therefore outer edges are calculated through image processing techniques.

These two types of boundary lines are called as *feature lines*, which will form together a three-dimensional *skeleton* model, reflecting the main characteristics of the roof's shape. As illustrated in Figure 1.1, the final three-dimensional model is made of several, approximately planar shaped polygon meshes built by triangulating the endpoints of the roof segments' feature lines.

To obtain a more realistic model, further correction modules are designed especially for handling concave triangulation problems described in Chapter 6.

Concave shaped roof segments appear frequently, however several well established triangulation methods, such as the used Delanuay triangulation[3] provided by CGAL [38], generate triangle meshes on the whole convex hull of the given points[4]. Consequently, as shown in Figure 1.2, irregularities on the roof sections' boundaries will be filled in concealing the rough edges.

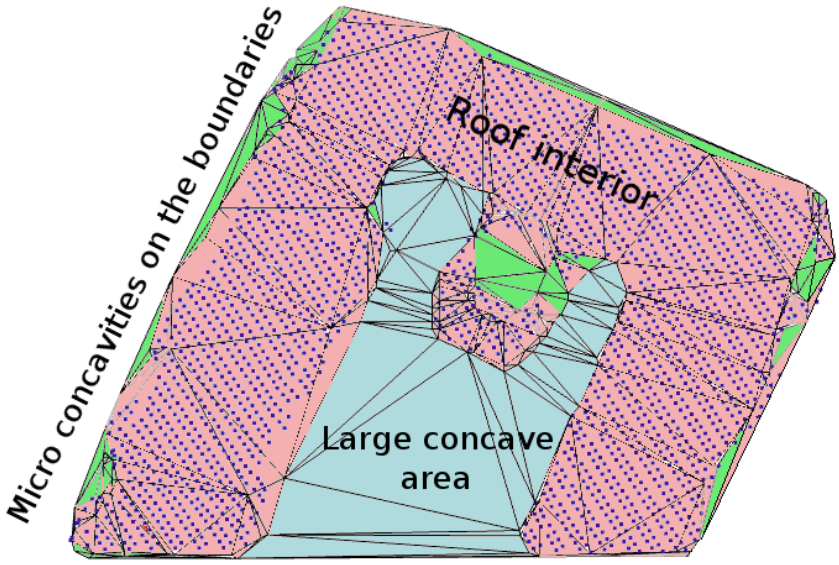


Figure 1.2. Demonstration of the convex hull problem as a result of Delanuay triangulation. The blue dots represent the points of this roof segment. The light magenta areas describe the true concave shape of the roof component, in which all triangles have at least one point in their interior. Green triangles do not contain any points but they are proposed to take part in the segment's polygon approximation, in order to fill in the micro concavities (holes, hollow parts) alongside the rough edges. The whole colored area highlights the convex hull of the shapes, where turquoise triangles form the *concave bay* (large concave area) needed to be eliminated from the final mesh.

At the same time, large concave areas (the turquoise field in Figure 1.2), which constitute important architectural features of the building, will also be filled in. This second effect means a severe data loss, therefore, I designed a procedure (detailed in Section 6.2) in which triangles lying on the concave parts of the mesh will be erased preserving the smooth boundaries. The procedure is based on a probabilistic graphical model described in Section 2.1.

[3]described in detail by Gallier et al. [31] in Section 8.3 *Delaunay Triangulations*

[4]"there is an intimate relationship between convex hulls and Delaunay triangulations", pronounced by Gallier et at. [31] in Section 8.4 *Delaunay Triangulations and Convex Hulls*

2. Used mathematical tools and algorithms

2.1 Markov Random Field

A Markov Random Field (MRF) is probabilistic graphical model, consisting of a set of random variables and a Markovian dependency model described by an undirected graph[1]. A Markov Random Field resembles a Bayesian network[2] in its representation of dependencies, but the main differences are that Bayesian networks are directed and acyclic, whereas Markov networks are undirected and may be cyclic.

MRFs are often used to model images, point clouds, triangle meshes because all these objects can be interpreted as a regular or irregular undirected graph[3] in which Markov properties can be assumed. Markov properties[4] ensure that the random variable assigned to a vertex depends only on random variables assigned to neighboring vertices. In other words two vertices are linked to each other only if the distance[5] between them is small enough.

2.1.1 Definition[6]

Let S be a finite set representing the vertices of an undirected graph $G = (S, E)$. Let $\Omega = (\Omega_s)_{s \in S}$ be a family of random variables defined on the set S, in which each random variable Ω_s takes a value $\omega_s \in \mathcal{L}$, which means that every site $s \in S$ is assigned variable Ω_s. The family Ω is called a random field.

We use the notation $\Omega_s = \omega_s$ to denote the event that Ω_s takes the value from a preliminary defined domain ω_s and the notation $(\Omega_{s_1} = \omega_{s_1}, \ldots, \Omega_{s_m} = \omega_{s_m}),\ s_i \in S$ to denote the joint event. For simplicity, a joint event is abbreviated as $\Omega = \omega$, where $\omega = \{\omega_{s_1}, \ldots, \omega_{s_m}\}$ is a *configuration* of Ω corresponding to a realization of the field. For a discrete label set $\mathcal{L}$, the probability that random variable Ω_s takes the value ω_s is denoted $P(\Omega_s = \omega_s)$, abbreviated $P(\omega_s)$ unless there is a need to elaborate the expressions. The joint probability $P(\Omega_{s_1} = \omega_{s_1}, ..., \Omega_{s_m} = \omega_{s_m})$ is denoted $P(\Omega = \omega)$ and abbreviated $P(\omega)$.

[1] for more information see [21], in Chapter Markov fields on graphs, pgs 24-34

[2] Stan Z. Li et al. [20] in Section 2.12 *Graphical Models: MRF's versus Bayesian Networks*

[3] Stan Z. Li et. al [20] on pg 23., Lavoué et al. [19]

[4] described by Chapman et al. [23] in Section 2.2.2 *Markov properties of GMRFs*, Stan Z. Li et al [20] in Section 2.10 *Strong MRF Model*,

[5] in our case Euclidean distance

[6] definition given by Stan Z. Li et. al [20], pgs 24-26

Let us generate a neighboring system $\mathcal{N} = \{\mathcal{N}_s \,|\, s \in S\}$ from the undirected graph $G = (S, E)$, so that $s \in \mathcal{N}_r$, $r \in \mathcal{N}_s$ and $\{s, r\} \in \mathcal{N}$ if and only if $s, r \in S$ and $(s, r) \in E$.

Definition. *Ω is said to be a Markov Random Field on S with respect to G if and only if the following two conditions are satisfied:*

$$P(\omega) > 0, \quad \forall \omega \in \Omega \qquad \textit{(positivity)} \tag{2.1}$$

$$P(\omega_s \,|\, \omega_{S \setminus \{s\}}) = P(\omega_s \,|\, \omega_{\mathcal{N}_s}) \qquad \textit{(Markov property)} \tag{2.2}$$

where $\omega_{S \setminus \{s\}}$ denotes the set of labels at the sites in $S \setminus \{s\}$, and $\omega_{\mathcal{N}_s} = \{\omega_s \,|\, s \in \mathcal{N}_s\}$ stands for the set of labels at the sites neighboring s.

Condition 2.2 means that an Ω_s random variable is conditionally independent of all other non-adjacent variables given its neighbors. In other words:

$$\Omega_s \perp\!\!\!\perp \Omega_r \mid \Omega_{\mathcal{N}_s} \quad \forall s \in S, \;\; r \in S \setminus \{i\} \setminus \mathcal{N}_s$$

As a consequence, the following conditions also hold:

Pairwise Markov property: Any two non-adjacent variables are conditionally independent given all other variables:

$$\Omega_s \perp\!\!\!\perp \Omega_r \mid \Omega_{S \setminus \{s,r\}} \quad \forall s, r \in S, \quad (s, r) \in E$$

Global Markov property: Any two subsets of variables are conditionally independent given a separating subset:

$$\Omega_A \perp\!\!\!\perp \Omega_B \mid \Omega_C, \quad A, B, C \subset S$$

where every path from a node in A to a node in B passes through C

2.1.2 Segmentation with MRF

Markov Random Fields are very useful in classification problems, especially in image and video processing, 3D point cloud and triangle mesh labeling due to the fact that many of these problems can be handled by graph based approaches[7].

In fact, a classification problem consists in determining the *optimal realization* of the field Ω. The optimal realization can be described by the label mask $\omega_{opt} = \{\omega_s \,|\, s \in S\}$ whose probability meets global maximum of the previously defined probability function, by using various observed features $x = \{x_s \,|\, s \in S\}$ about the graph G. If the space of all possible configurations is denoted as $\Gamma = \mathcal{L}^{|S|}$, the optimal labeling can be obtained as

$$\omega_{opt} = \arg\max_{\omega \in \Gamma} P(\omega \,|\, x)$$

[7] Stan Z. Li et al [20] in Section 1.1 *Labeling for Image Analysis* (image labeling), Rui Hu et al. [29] (video processing), Lavoué et al. [19] (3D mesh segmentation), Börcs et al. [2,3] (point cloud classification)

Applying the Bayes' rule we get:

$$\begin{aligned}\omega_{opt} = \arg\max_{\omega\in\Gamma} \frac{p(x\,|\,\omega)P(\omega)}{p(x)} &= \arg\max_{\omega\in\Gamma}\ p(x\,|\,\omega)P(\omega) \\ &= \arg\min_{\omega\in\Gamma}\ \Big(\ln p(x\,|\,\omega)^{-1} - \ln P(\omega)\Big)\end{aligned} \tag{2.3}$$

Prior model

The joint probability $P(\omega)$ constitutes the ***a priori*** knowledge on the model (i.e. knowledge before the observation) . In 1971, Hammersley and Clifford theoretically proved that Markov Random Fields show to be equivalent to Gibbs distributions [24]. Consequently, the join probability $P(\omega)$ can be modeled by a Gibbs distribution:

$$P(\omega) = Z^{-1} \times e^{-\frac{1}{T}\cdot\lambda E_s(\omega)}$$

where

$$Z = \sum_{\omega\in\Omega} e^{-\frac{1}{T}\cdot\lambda E_s(\omega)}$$

is a normalizing constant, λ is a smoothness coefficient, T is a temperature factor being assumed to be 1 for simplicity. $E_s(\omega)$, henceforth denoted as E_s[8], is the so called ***smoothness energy***, a kind of measure of the classification's smoothness, favoring homogeneous regions. With reference to Kolmogorov et al. [25], the *smoothness energy* is defined as follows:

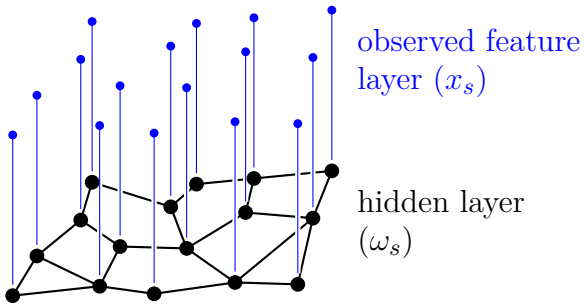


Figure 2.1. Inordered dependency graph of a Markov Random Field. The small blue nodes are observed, the black nodes are hidden.

$$E_s = \sum_{\{s,r\}\in\mathcal{N}} V(\omega_s, \omega_r) \tag{2.4}$$

where V is a user defined cost function, hence called ***smoothness cost***. Let $V(i,j)$ be equivalent with the Kronecker delta $\delta(i,j) = \delta(i-j) = 1$ if and only if $i = j$.

Observation model

The ***observation*** or ***data model*** represents the knowledge of some kinds of indirect informations about the field's realization, which is given by the following probability:

$$p(x\,|\,\omega) = \prod_{s\in S} p(x_s\,|\,\omega_s) \tag{2.5}$$

Considering the negative logarithm of 2.5 we get:

$$\ln p(x\,|\,\omega)^{-1} = \sum_{s\in S} \underbrace{\ln p(x_s\,|\,\omega_s)^{-1}}_{d(x_s\ |\ \omega_s)} = E_d \tag{2.6}$$

[8] "s" stands for *smoothness*, not to be confused with $s \in S$

where x_s values constitute the ***observed features*** on each s vertex (eg. pixel values of an image), $d(x_s \mid \omega_s)$ are called ***data costs*** or ***data terms***, and their sum over the set S gives the ***data energy*** (E_d).

Substituting equation (2.6) into equation (2.3) we get the final energy function, which should be optimized to obtain the best ω_{opt} global labeling:

$$\omega_{opt} = \arg\min_{\omega \in \Gamma} \left(\sum_{s \in S} d(x_s \mid \omega_s) + \lambda \sum_{\{p,q\} \in \mathcal{N}} \delta(\omega_s, \omega_r) \right) = \arg\min_{\omega \in \Gamma} (E_d + \lambda E_s) \tag{2.7}$$

To conclude, the optimal labeling is a configuration which minimizes the energy $E = E_d + \lambda E_s$. Further challenges rest upon defining the $d(x_s \mid \omega_s)$ *data costs*, but this problem is very task specific, therefore I will detail this issue in Section 6.2 in which I have used this method for a triangle mesh classification.

A typical example for using MRF models is foreground-background segmentation of a grayscale image. Here the pixels correspond to the vertices of the graph (S). The edges of the graph (E) are represented by the neighboring pixel pairs, consequently every pixels' label (whether belongs to the background or foreground) depends on their neighboring pixels' label.

Since $\omega_{opt} \in \Gamma$, the number of possible realizations is $|\mathcal{L}|^{|S|}$, thus finding the global optimum is a non-polynomial problem. However many relaxation algorithms (simulated annealing, iterated conditional modes, etc.) are devoted to find the optimal or at least an efficient sub-optimal solution in an efficient way.

To find the quasi global minimum of the energy function we used the graph cuts based optimization technique developed by Olga Veksler, using the libraries provided by Yuri Boykov and Vladimir Kolmogorov [25–28].

3. Point cloud classification

In this section I will present the initial classification method based on the work of Börcs and Horváth et al. [1] for the Students' Scientific Competition and the method introduced by Lafarge et al. [4, 5] and Zhou et al. [13].

A point cloud is a set of three dimensional points, which are described by their (x, y, z) coordinates. Besides its coordinates, each point is assigned further parameters provided by the aerial LiDAR scanner. One of these parameters constitutes the *number of returns* of the radiated laser ray (n_r) that yields information about the object's surface. Each return is stored as a separate point and it is labeled with the ordinal number ($k < n_r$) of its arrival, representing the second mentioned parameter, the *return number*.

As we have mentioned in Section 1.1, when multiple returns occur, the first reflections belong to objects in higher altitudes with irregularities like tree canopy ($n_r > 1$ and $k \neq n$). By means of these two attributes, vegetation could be distinguished beside clutter. Vegetation and clutter make up the so-called *outliers*. Clutter regions consists of sparse cloud sections constituting vertical objects like building walls, and characteristic lines, so called *power lines* like building edges or power cables in higher altitudes.

Since vegetation constitutes an important part of the reconstruction, we have to extract vegetation from the outliers. First, sparse cloud section of clutter can easily be excluded, considering the number of neighboring points within a given radius (R). However building walls often contain small oblong cloud pieces below each other representing the shoulders of the vertical walls at the level of the different buildings floors. These oblong segments, and other power lines differs from tree canopies in their scatter characteristics. Consequently vegetation can be excluded considering the cloud's local covariance.

Inlier points are well-structured dense cloud sections containing approximately horizontal regular objects with large area such as roofs, ground

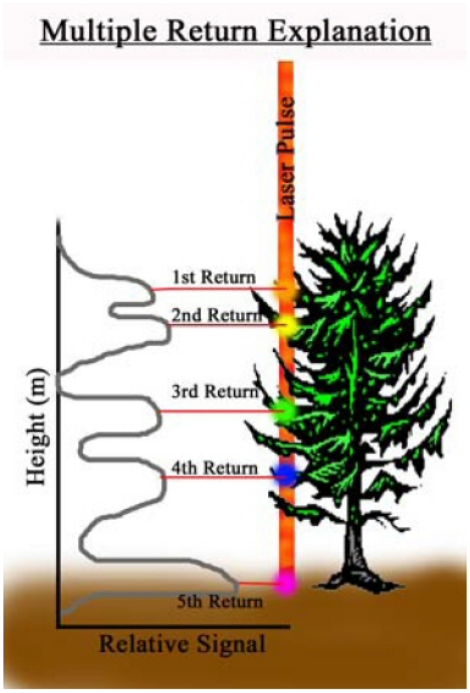


Figure 3.1. In this case of multiple return, a single laser ray will produces 5 returns resulting 5 points. The $n_r = 5$ for all 5 points, while $k = 1, 2, 3, 4, 5$ is the ordinal number of their return.

or larger vehicles. After we have extracted the ground points using the random sample consensus (RanSaC)[1] optimal plane fitting method, we are able to calculate the elevation relatively to the ground. Objects close to the ground level are likely to be vehicles which is out of our interest, therefore we concentrate on higher level cloud sections.

3.1 Extracting connected roof components

In this step we separate connected building blocks in order to reduce the roof segmentation's complexity and increase its robustness exploiting local regularities. To extract building blocks, Lafarge et al. [4] used image processing algorithms analysing the cloud's planimetric projection, however this task does not require long processing time even if we do it in the three-dimensional space.

Therefore, we use a 3D region-growing (flood fill) algorithm starting from an arbitrary point in the roof cloud. After the flood fill stops we extract the retrieved cloud segment, and we start again with a new arbitrary seed point until the remaining cloud becomes empty. This procedure results in several point clouds, where each contain a single roof section. These components are illustrated in a common cloud with different coloring in Figure 3.2. In order to obtain a reasonable processing time we used a kd-tree structure[2], so that the neighborhood searching algorithm operates in $\mathcal{O}(n \log n)$ fashion. As a result, we obtain point clouds containing a reduced number of points.

Figure 3.2. Each connected roof compenent is illustrated with a specific color

In fact, this task can also belong to the point cloud classification module considering its functionality. We will apply a further filtering operation, which extracts small components, like insignificant building fragments, remained cloud pieces from the previous classification (eg. not recognized tree canopies).

More specifically, the core algorithm is processed several times on each building block whose measurement data set is smaller than 50000 points but larger than a given value, in order to avoid noise reconstruction.

[1] a detailed description is available by Börcs and Horváth et al. [1]

[2] implemented by Point Cloud Library et al. [37] (`http://www.pointclouds.org/`)

4. Roof segmentation

In this section we are going to extract planar roof segments on the basis of their orientation. First of all, we estimate a surface normal at every point of the roof cloud using the Point Cloud Library's [37] implementation of Moving Least Squares (MLS) algorithm (Figure 4.1a). Since we know every point's normal, we apply a clustering algorithm to detect the representative directions in which the planar roof components face (black vectors in Figure 4.1b). These few directions will represent separate clusters with different labels. As Figure 4.1c illustrates, every point will be assigned an appropriate label (i.e. color), depending the point's normal. As a result, the points of every roof segment having similar orientation will be given the same label. Afterwards, a region-growing is applied on the cloud knowing the labels that the roof points belong to. In Section 4.5, we are going to apply a smoothing algorithm to eliminate insignificant, small segments and other noises generated by region-growing (Figure 4.1c). Consequently, every planar continuous roof component will be distinguished by a unique segment ID, and then we will be ready to perform the polygon approximation for every roof segment, which step is detailed in the next chapter.

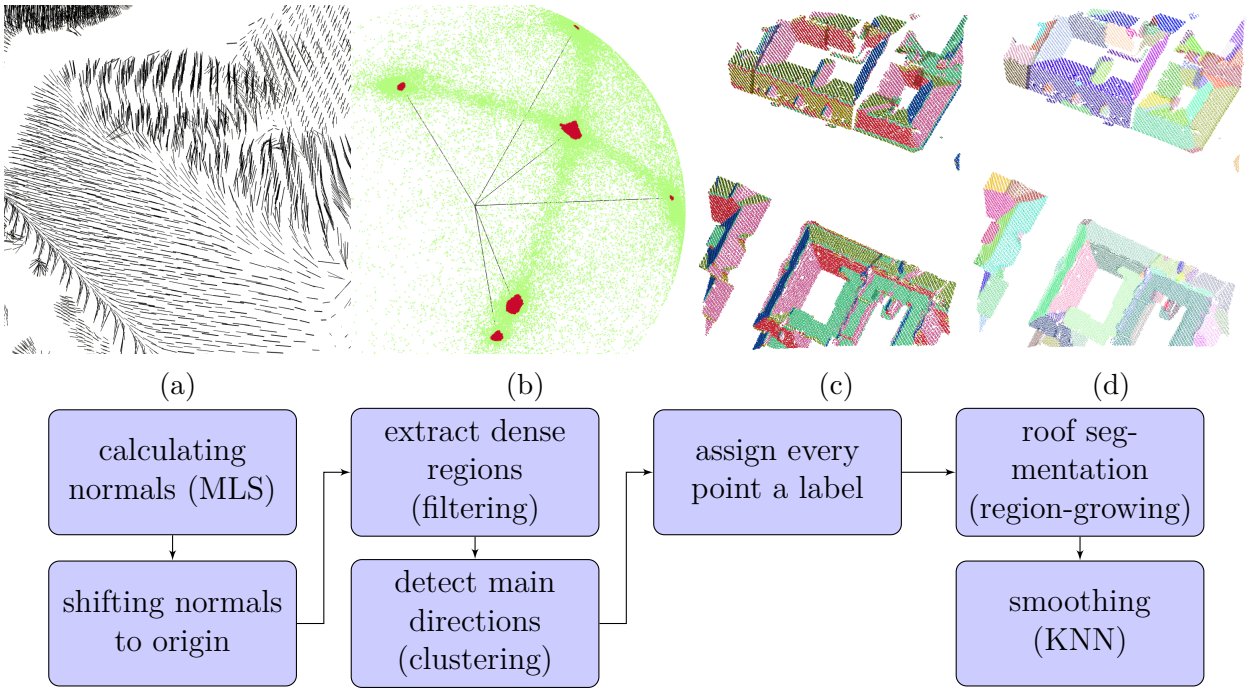


Figure 4.1. Workflow of roof segmentation.

4.1 Shifting normals into origin

As we have mentioned in this chapter's introduction, the roof classification's assembly commences with an MLS algorithm that estimates a surface normal at each point through analysing its neighborhood.

Henceforth, every p point will have six parameters: the $\vec{r}_p = (x_p, y_p, z_p)$ coordinates, describing its position, and $\vec{n}_p = (n_{xp}, n_{yp}, n_{zp})$ coordinates of its normal vector, where $\|\vec{n}_p\| = 1$. If we consider $\vec{n}_p$ as a three-dimensional $N_p(n_{xp}, n_{yp}, n_{zp})$ point, N_p is located on the surface of the unit sphere, since $\|\vec{n}_p\| = 1$. Then we store each N_p point in a new cloud obtaining a spherical, origin centered cloud, which is shown in Figure 4.1b.

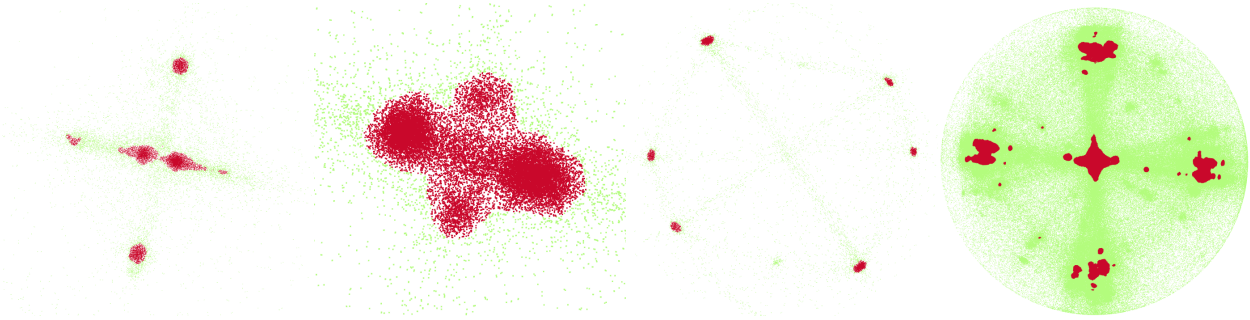
To rephrase, we shift every unit length normal to the origin and we consider their endpoints which together consolidate a cloud around the unit sphere. This cloud can provide us later useful global statistical information about the normal vectors over the whole scene.

In the following step, we remove sparse cloud sections from the spherical cloud. The remained dense regions, shown in Figure 4.1b, correspond to the dominant normal directions of the original point cloud, which may indicate different planar surface components.

4.2 Clustering

A *cluster* is a group of similar elements gathered or occurring closely together. In our case, a cluster is defined as a set of points located very close to each other being arranged within a relatively small area. Furthermore, we introduce the concept of *cluster point*, which represents a particular cluster, and it is determined as the arithmetic mean position of all the points in the same cluster.

In order to retain the main directions, we estimate the cluster points from the dense regions of the spherical cloud. The task is not trivial in view of the wide range of types of the occurring normals' cloud, which are demonstrated in Figure 4.2. According to a human observer, the single red cloud section in Figure 4.2b can be considered as a single cluster but also as two, or even four overlapping clusters. On the other hand, the cloud in Figure 4.2d features five significant directions, but in addition, small dense sections appear there which can also be presumed to be separate clusters.



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

Figure 4.2. A few examples of normals' clouds. The green points constitute the original normals' cloud, from which dense regions, being colored red, are extracted. The filtering consists in removing every point having less neighbors within a given radius R than a given threshold K. The filtering algorithm is adaptive in a sense that R is inversely and K is directly proportional to the size of the cloud (i.e. number of points in the cloud).

In order to detect cluster points, we will follow two different strategies. First, we will proceed an algorithm which operates on the normals' cloud in the three-dimensional space, described in Section 4.2.1, in which we will try to extract from all dense regions at least one separate cluster even if this will result in several needless cluster points. As for our second approach, we project the filtered normals' cloud onto the xy plane, as proposed in Section 4.2.2. Then we use morphological operations on binary images to find as few cluster points as possible.

4.2.1 Finding cluster points in 3D

This step is an iterative algorithm which processes on the filtered cloud of normals. In the initial stage, we have no detected cluster points, and every normal point is *independent* in the sense that they have not been given any label yet (i.e. do not belong to any of the existent clusters). Then the algorithm iteratively generates new cluster points with new labels. In a single iteration, we choose randomly an independent point from the filtered cloud of normals. The chosen point will be a new cluster point introducing a new label, which will be assigned to all independent points in the new cluster point's neighborhood within a given radius called *deviation*. The algorithm repeats these steps until no independent points are found. A pseudocode of the proposed clustering algorithm is shown below.

```
function clustering_in_3D (normals_cloud)

    cluster_points = empty_list;

    % number of independent points
    count = size(normals_cloud);

    while count > 0
        new_point = a random point from normals_cloud;
        new_label = a random integer from 1 to 16777215 (0xffffff);

        cluster_point.append(new_point);

        for p in normals_cloud
            if (p.label = null) and dist(p, new_point)
                p.label = new_label;

                % updating the number of independent points
                count = count - 1;
            end
        end
    end

    return cluster_points
end
```

4.2.2 Finding cluster points by projection

First of all, the cloud of the normals is projected onto the xy-plane in the following way: if a point's projection hits a cell on the 2D grid, that cell will be given value 1, otherwise value 0. As Figure 4.3e demonstrates, this projection returns a binary image containing black spots (objects) corresponding to the dense cloud sections of normals' cloud in Figure 4.3d.

As discussed earlier, our objective is to detect as few cluster point as possible, therefore, we eliminate small black objects using a few morphological operations, as illustrated in Figure 4.4. Then a morphological shrinking[1] is adopted to obtain each black object's centroid. The challenge is that on the boundaries (near the unit sphere's "equator" at $z = 0$, see Figure 4.3a) the shapes of the objects deforms significantly during the projection. In order to avoid this pitfall, we transform these (x, y, z) vectors to a spherical (ϑ, φ, r) coordinate system, where r marks the point's distance from the origin, φ points the angle between the z-axis and the plane's normal, respectively ϑ indicates the normal's orientation on the xy-plane. Next we shrink this spherical cloud around the sphere's north pole (i.e. we decrease the value of φ by scaling it). Since $r = \sqrt{x^2 + y^2 + z^2} = 1$, the transform should look like the following:

$$\begin{cases} \varphi = \arccos(z) \\ \vartheta = \operatorname{sgn}(y)\ \arccos\left(\dfrac{x}{\sin\varphi}\right) \\ \varphi = \dfrac{\varphi}{A}, \end{cases} \tag{4.1}$$

where $A \in (1, \infty)$ is a scaling factor. The inverse transform into Cartesian space lends its shape:

$$\begin{cases} K = \left(\sin\dfrac{\pi}{2A}\right)^{-1} \\ x = K \sin\varphi\ \cos\vartheta \\ y = K \sin\varphi\ \sin\vartheta \\ z = \cos\varphi, \end{cases} \tag{4.2}$$

where K is a normalization factor, that scales the spherical sector from region

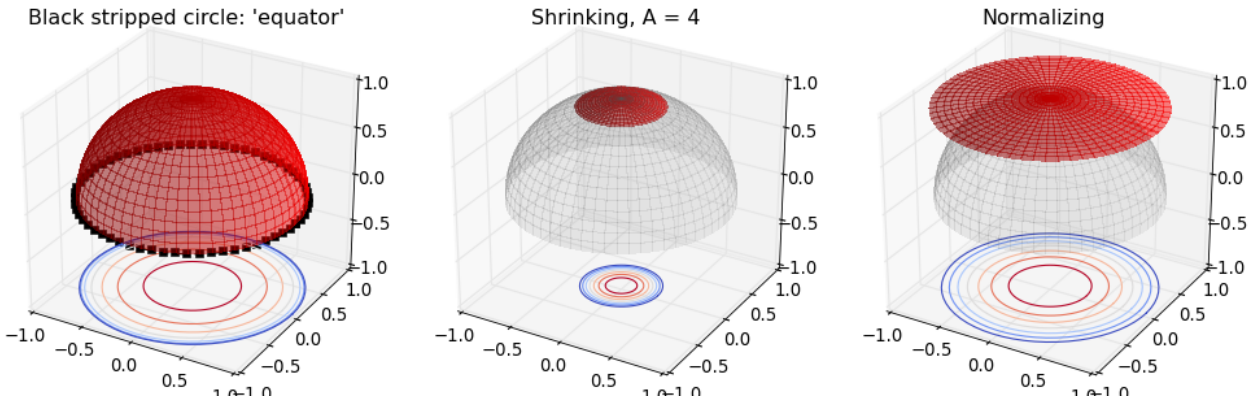
$$\left[-\sin\frac{\pi}{2A}, \sin\frac{\pi}{2A}\right] \times \left[-\sin\frac{\pi}{2A}, \sin\frac{\pi}{2A}\right] \times \left[\cos\frac{\pi}{2A}, 1\right].$$

into

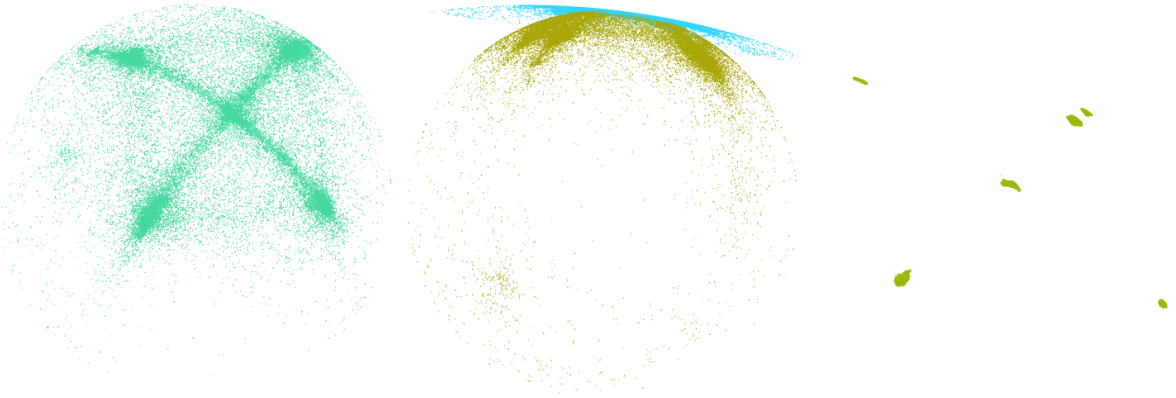
$$[-1, 1] \times [-1, 1] \times \left[\cos\frac{\pi}{2A}, 1\right]$$

The obtained cloud of normals can easily projected. After the morphological shrinking, we elevate and transform the remained black points into Cartesian space. Finally the cluster points are retained.

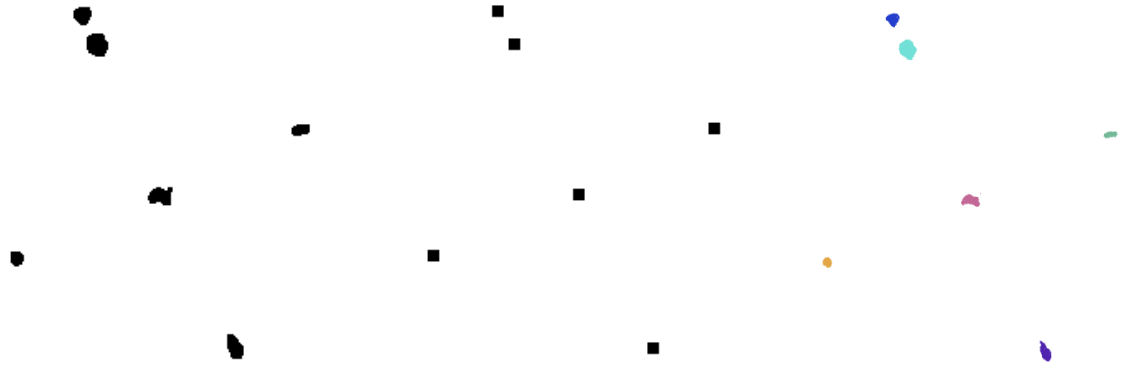
[1] More description available et al [16], pgs 411-414



(a) The proposed transform, which makes the cloud of normals more flat. The original shape of the cloud (left) is shrinked adopting the transform in eq. 4.1 (middle), and scaled using the inverse transform in eq. 4.2 (right). We can compare the contour plots of the first and third mesh. In the first contour plot, the concentric circles are lying near the "equator" with a high density. On the third plot the circles are lying in an approximately uniform distribution around the origin.

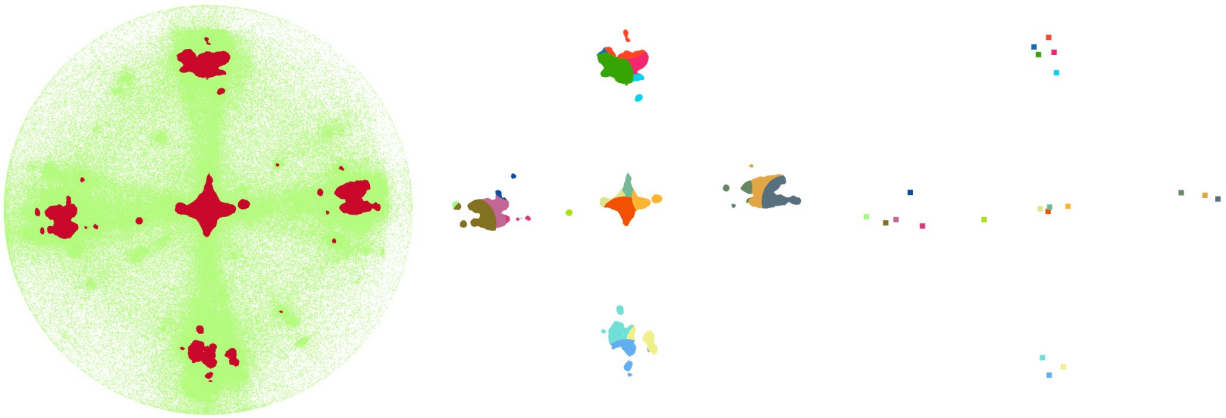


(b) normals' cloud (c) transformed cloud (red) (d) dense regions

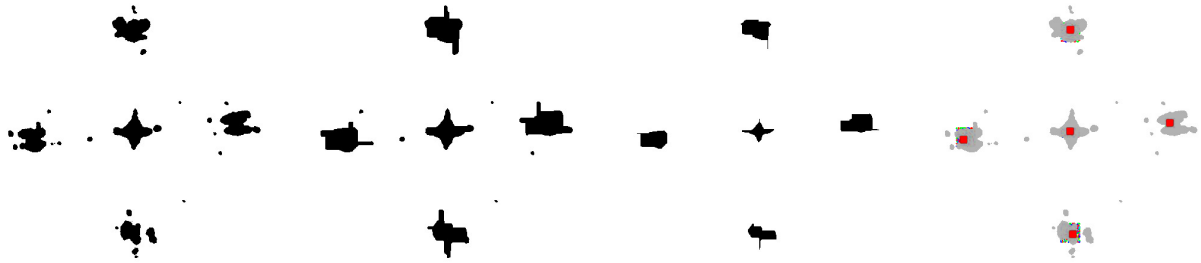


(e) projection (f) morphological operations (g) clusters in 3D

Figure 4.3. A detailed illustration of clustering.



(a) The clustering algorithm, which processes in 3D, has detected 21 cluster points.



(b) After the projection we are able to operate on the image, so that the irrelevant objects disappear. The first figure constitutes the binary image of the cloud's projection in which five large objects can be found surrounded by smaller spots. We generate a morphological closing to merge these small spots into the large black areas (second image). Afterwards, we proceed a morphological erosion to eliminate the remained irrelevant black items (third image). The previous step necessity depends on the projection of the filtered cloud, because if the dense regions corresponding to the main directions are relatively small, this step will erase the respective black objects. Finally, a morphological shrinking is adopted which shrinks each black object into a single pixel which are highlighted in the last figure. In contrast to our first approach, this method correctly detects only 5 cluster points corresponding to the five main directions.

Figure 4.4

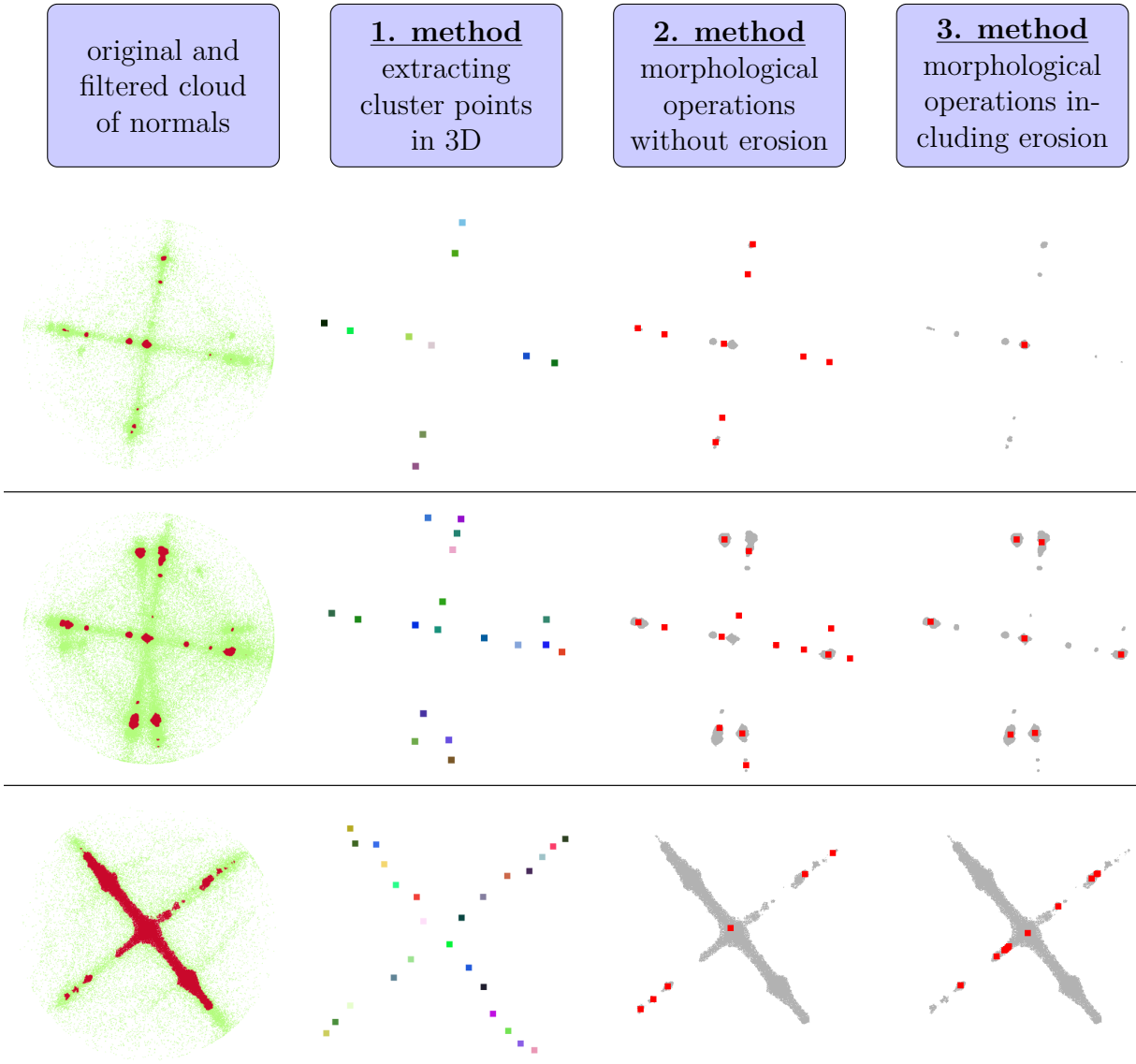


Figure 4.5. A comparative study of the two different clustering techniques on further examples. The first column presents the original (green) and the filtered normals' cloud (red). The second column represents the cluster points calculated in 3D. The third and fourth columns illustrate the cluster points determined by the binary morphological operations. The difference between the last two columns is that in the third column we missed out the step of morphological erosion. In case of the sample in the first row, the morphological erosion eliminates a few objects corresponding to the significant directions, therefore the second method proved to be the best choice. On the other hand, in case of the second sample, too many cluster points are found without erosion, hence we chose method no. 3. Concerning the last sample, we cannot use shrinking because the large, continuous oblong shaped object will be eroded into a single pixel representing a single cluster, so we use the first method here.

4.3 Roof labeling

In the following, our task is to assign a proper label to each roof point considering the surface's normal vector at the respective point. However, there are cases (Figure 4.5 - third example) when several cluster points are extracted from a single dense region of the normals' cloud, which can cause problems. In particular, if we assign each roof point the nearest cluster point's label, the points of a certain roof component with a specific orientation can belong to different labels, due to the fact that small differences in direction can occur even among points of the same segment. To handle the problem, we designed a simple procedure, in which every point will be assigned the label of the first found cluster point, whose distance from the respective roof point is smaller than a given threshold, called *tolerance*. The distance is Euclidean and is calculated in the following way:

$$d = \sqrt{(n_{x_p} - x_l)^2 + (n_{y_p} - y_l)^2 + (n_{z_p} - z_l)^2},$$

where $\vec{n}_p = (n_{xp}, n_{yp}, n_{zp})$ is the normalized normal vector and $\vec{p}_l = (x_l, y_l, z_l)$ is the cluster point belonging to label l. If the *tolerance* is high enough, the roof points with small differences in their corresponding normals will be given the same label. The main aspects of the algorithm are presented in the following pseudocode.

```
function roof_labeling(cloud, cluster_points, tolerance)
    for each point in cloud do
        min_dist = inf
        min_label = 0

        -- itereting through all possible cluster points
        for each cluster_point in cluster_points do
            dist = distance(point.normal_xyz, cluster_point.xyz)

            -- if the distance is less then the tolerance,
            -- the cluster point has already been decided,
            -- even thought there could be closer cluster points
            if dist < tolerance then
                point.label = cluster_point.label
                min_label = 0
                break
            end

            -- in case when no close cluster point is found,
            -- we store the label of the nearest cluster point
            if dist < min_dist then
                min_dist = dist
                min_label = cluster_point.label
            end
        end

        -- if no close cluster point was found, the actual point
        -- is assigned the label of the nearest cluster point
        if (min_label != 0) then
            point.label = min_label;
        end
    end
end
```

Figure 4.6. Pseudocode of the roof clustering presuming the cluster points. Parameter `cloud` represents the filtered normals' cloud

4.4 Segmentation

The segmentation itself is a conditioned flood-fill process in which we are looking for continuous cloud sections with points assigned the same label. Let us call these continuous cloud sections ***roof components*** or ***roof segments***. Since the points belonging to a certain roof component have the same label (i.e. their corresponding normals represent the same orientation), the roof segments are approximately planar shaped cloud segments, henceforth we will often model them by their approximated plane equation and centroid, especially in geometrical calculations. A roof component's centroid constitutes the arithmetic mean of its points, and its optimally (in least mean square fashion) estimated plane equation is determined by Principal Components Analysis (PCA). To distinguish the roof segments we will assign each component a unique ID (represented by a color value in Figure 4.7)

4.5 Correction by local features

The segmentation produces a slight noisy result. The problem is that small roof segments are also detected, since we have not yet prescribed any minimum requirements concerning their size (number of points contained). In view of the fact that a specific point's label depends mainly on it's neighbors (i.e. two points situated far from each other are to be considered conditionally independent points), the noise could be reduced using K-nearest neighbors (KNN) algorithm. According to Lavoué et al. [19] the previously mentioned algorithm could be replaced by a smoothing algorithm based on MRF in order to improve the correction's performance. Since, the above mentioned KNN performs sufficient well in our considered test data, we decided to schedule the implementation of Lavoué's technique et al. [19] only for future work.

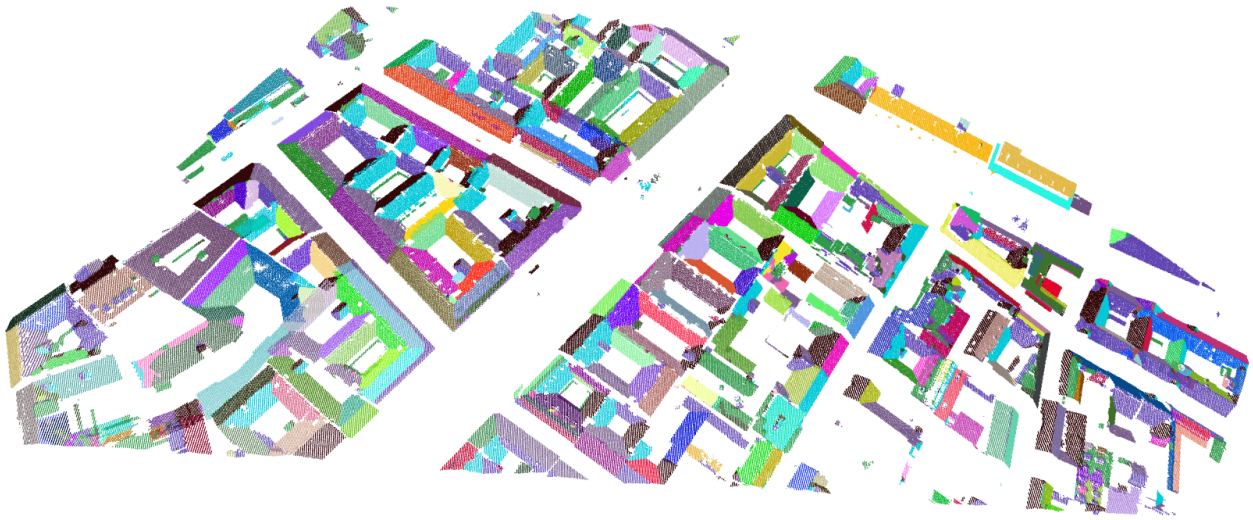


Figure 4.7. Roof segmentation output in a relatively large point cloud. The separate colors mean different IDs. We can see, that the great majority of the corresponding roof segments are colored with a single color (their points are given the same unique ID)

5. Reconstruction

This system part implements a key element of the workflow, because here we will generate a complete 3D model from the cloud regions segmented as roof. The reconstruction consists in triangulating every planar roof segment using the detected boundary lines.

5.1 Generating 3D skeleton

In this section I aim to present my approach of generating a 3D skeleton model from the previously segmented roofs cloud. First, *inner* edges will be computed, then a projection is adopted intending to detect *outer* boundary lines using OpenCV's Canny edge detector. The edge detector identifies points in a digital image at which the pixel values changes sharply. The detected points are marked black in the output image having white background. After elevating the black points of the Canny edge image we retain an edge cloud that will need a further tangent estimation considering each point's neighbors. Knowing the tangent vectors of the boundary points, a line fitting step is proceeded in a straightforward way using a common principal component analysis (PCA) technique.

5.1.1 Plane intersections

Before all, we have to compute the equations of all planes representing each roof segment. This can be fulfilled using PCA, that will fit a plane to the respective roof points in a least mean square (LMS) manner. Afterwards, inner edges are computed as the intersection of two neighboring planes. There are cases when we cannot rely on the intersection, in particular, if the two segments are nearly parallel. Therefore, we select points from the common boundary of the two segments and fit a line to them. This common line guarantees the model's watertightness, because it will take part in triangulation of both roof-segments, therefore the two polygons, representing the two roof segments, will interlock.

5.1.2 Outer edge detection

Outer edge points could be detected in various ways. For example, the Point Cloud Library provides a boundary detection procedure based on 3D techniques. Lafarge et al. [4,5] has also been chosen a 3D way to detect edges by minimization of the squared error. My approach, consists in projecting the cloud onto the xy plane generating a grayscale pixel grid (image). This mapping is also called as planimetric projection.

The value of each pixel in the projected image, constituting the projection of the corresponding point, is the linear transformation of the z value of the respective point. In cases when no points are projected into a specific pixel, its value remains 0 (ground level). Henceforward, we call this projected image as z-image.

As Figure 5.1 illustrates, the projection of the cloud does no cover the whole area on the image, therefore we fill in the white pixels using a median filter[1]. Afterwards, an edge detection is applied on the filtered z-image, in order to identify the edges lying between ground and roof points.

Our objection is that elevation differences below a given minimal threshold should not be marked as an edge, otherwise many spurious lines would be generated in the following steps, which would produce a highly complex and unmanageable mesh.

At the same time, each relevant outside wall should be detected. Consequently, we have to design a transform that scales the cloud, so that its projection underlines the difference between the ground level and roof level. To obtain this, the difference between the values of projections of the ground points and the lowest level roof points should be high enough for the sake of being noticed.

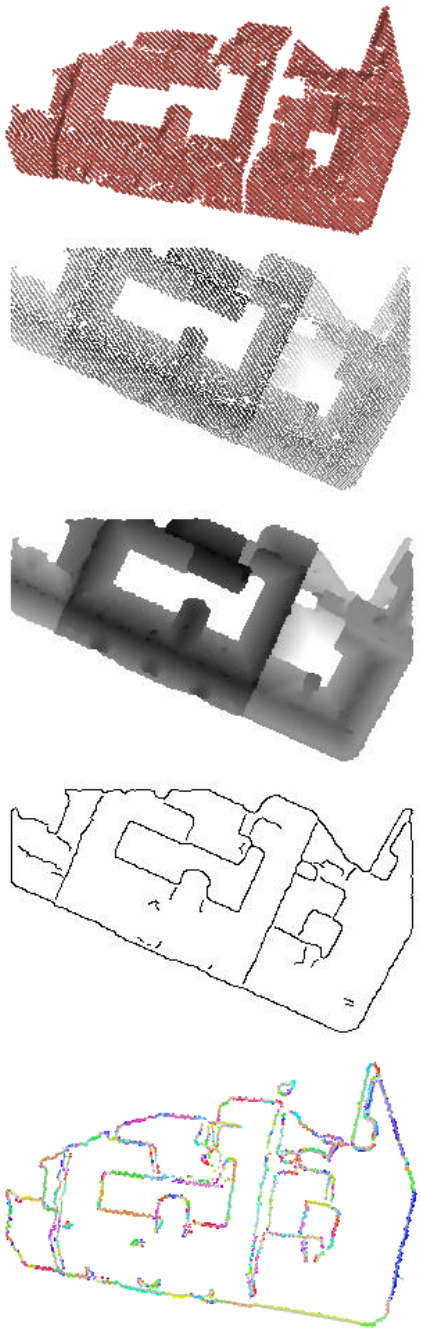


Figure 5.1. Outer edge detection's assembly - projection - median filtering - elevation into the 3D space and segment fitting.

Edge elevation

After the application of the Canny operator, we map the black pixels of the edge image to the real 3D space applying the inverse of the projection transform. In fact projection is not an invertible operator, but now the the elevation values are stored in the z-image. This will result a new edge cloud indicating the edge points of the roofs in the cloud. We compute the covariant matrix of the neighbors of each point of the elevated edge cloud to retrieve the eigenvector of the highest eigenvalue (the direction followed by the neighboring points). In other words, we need to know the tangent vectors of the cloud in each point of the cloud.

Figure 5.1 illustrates the workflow of the outer edge detection. The 1st figure illustrates the original 3D point cloud, which is projected onto the xy plane (2nd figure). We apply a median filtering on the z-image (3rd figure), then edge detection is applied (4th figure) which is elevated into the 3D space using the pixel values of the z-image. A

[1] a detailed descriptions is available at [16], pgs. 271-275

further segment fitting step is applied on the retained edge cloud resulting in the 3D *outer edges*, as illustrated in the 5th image of Figure 5.1.

5.1.3 Segment fitting by minimizing squared error

In the following we have to fit lines to the elevated edge points. Our first approach was to cluster the edge points by their three-dimensional direction. We used the same technique that we have mentioned as a solution for roof segmentation, apart from considering each point's tangent vector in stead of their normal's. This realization resembles a two dimensional Hough transform in performance. Long segments have been recognized well, but shorter ones have been punished. To avoid this conflict I have implemented a second approach that collects neighboring points whose tangents are close to each other (i.e. a conditioned flood-fill algorithm without an anticipatory clustering on their direction vectors).

5.2 Mesh generation from skeleton model

Knowing the edges of each roof segment, we loop over the roof segments and apply the triangulation one by one. Before triangulating, we transform the roof component, so that the plane fitted to it becomes equivalent to the xy plane, then we project outer edges onto the roof's plane (now xy plane). With these transforms the 3D triangulating problem has just been reduced to a 2D triangulation problem. The triangulation itself that we used was a non-constrained Delaunay triangulation applied on the endpoints of roof segment's edges. The implementation we used triangulates on the whole convex hull of the edge endpoints, which means, that the concave parts of the roofs are erroneously filled with false triangles. Therefore, I have developed an algorithm which analyzes the concavity of each roof segment and refines the polygon if the difference between the convex hull and the desired shape is relevant. Using the proposed improvement, even concave roof parts can be correctly reconstructed. Since handling the concave problem is a key contribution of my thesis, I dedicated the whole Chapter 6 to this issue.

6. Solutions for concave problem

In this chapter I propose a new method for two-dimensional concave polygon triangulation using a Markov Random Field. The algorithm refines an initially generated triangle mesh on the whole convex hull of the polygon by eliminating the invalid triangles appearing on the external parts of the shape. At the same time, the procedure preserves smooth boundaries of the final mesh even if the input polygon contains rough edges. Before introducing the proposed algorithm (Section 6.2), we also demonstrate the limitations of a publicly available solutions for the same problem, called PSLG (Section 6.1).

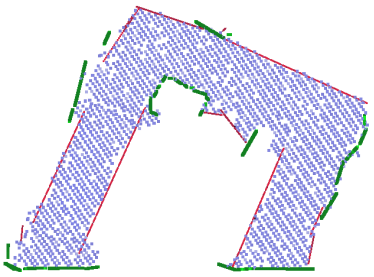


Figure 6.1. The input of the concave roof reconstruction - the point cloud (blue) of a concave roof segment end its *inner* (red) and *outer* (green) boundaries

The algorithms of this chapter are used for reconstructing the concave shaped roof components from their corresponding point cloud sections, which have been extracted by the algorithm proposed in Chapter 4, and from their boundary lines detected from the source cloud by the procedure detailed in Chapter 4. The illustration in Figure 6.1 highlights the input of our proposed concave roof reconstruction.

6.1 Concave problem with *Triangle* Software

Triangle Software provides a 2D polygon triangulation technique as a generic solution for the concave problem. *Triangle* is an open-source two-dimensional quality mesh generator and Delaunay triangulator, winner of the 2003 James Hardy Wilkinson Prize in Numerical Software[1].

Triangle can generate a constrained Delaunay triangulation (CDT) on a Planar Straight Line Graph (PSLG), which can be interpreted as a bounded, continuous 2D region with possibility of containing arbitrary number of holes and concavities. Furthermore, edges representing the PSLG's boundaries are allowed to intersect each other.

A constrained Delaunay triangulation operates within the bounded region of a PSLG, hence the concave parts at the outer boundary will not be triangulated. With the purpose of generating such kind of triangulation, first a PSLG has to be generated knowing the edges of the given roof segment. Assuming that our roof components do not contain holes we can interpret PSLG as an arbitrary polygon with a reasonable prospect that its edges intersects one another.

[1]prize awarded by Argonne National Laboratory, the National Physical Laboratory, and the Numerical Algorithms Group in honor of the outstanding contributions of Dr. James Hardy Wilkinson to the field of numerical software (`http://www.mcs.anl.gov/research/opportunities/wilkinsonprize/index.php`)

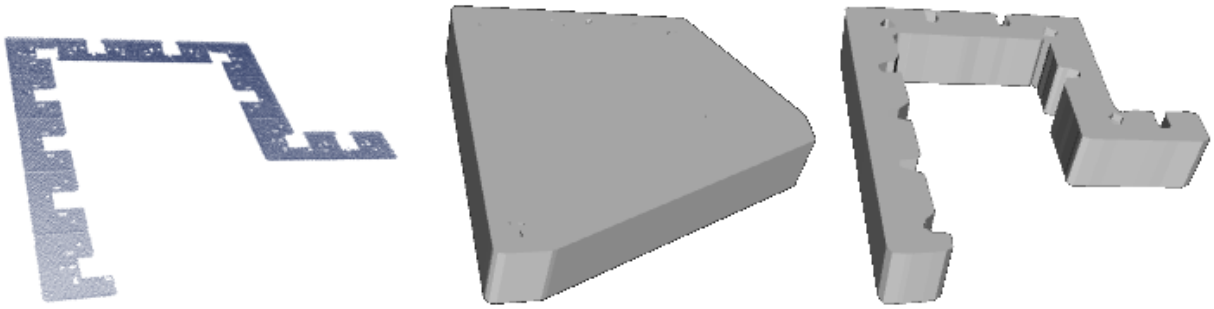
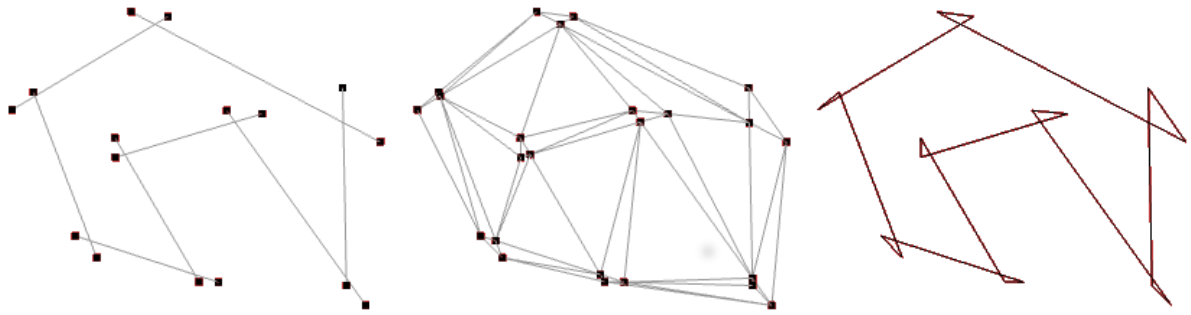
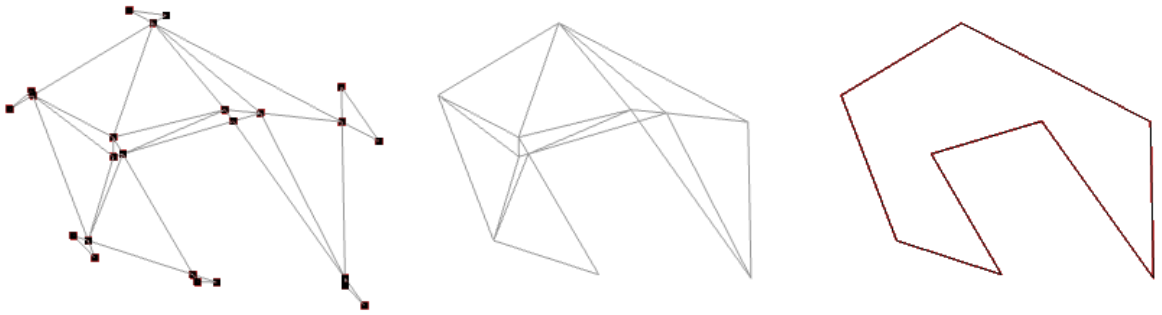


Figure 6.2. Concave problem, points of a concave roof segment (left), triangulation on the whole convex hull (middle), triangles on the concave part removed by an a stochastic algorithm described in Section 6.2 (right)

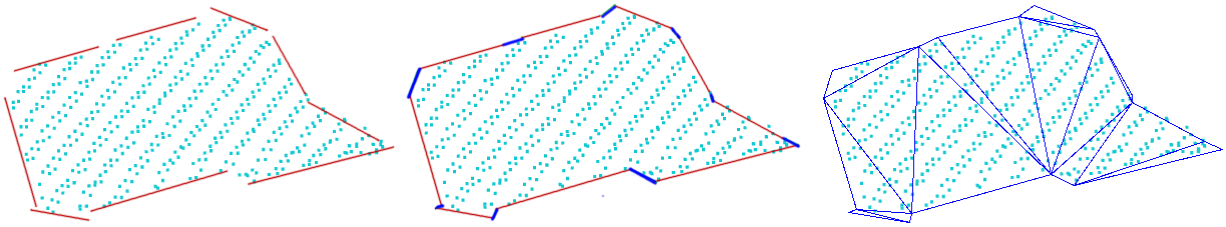


(a) original edges (b) triangulation on convex hull (c) generated polygon



(d) concave triangulation (e) eliminating bad triangles (f) outer sides

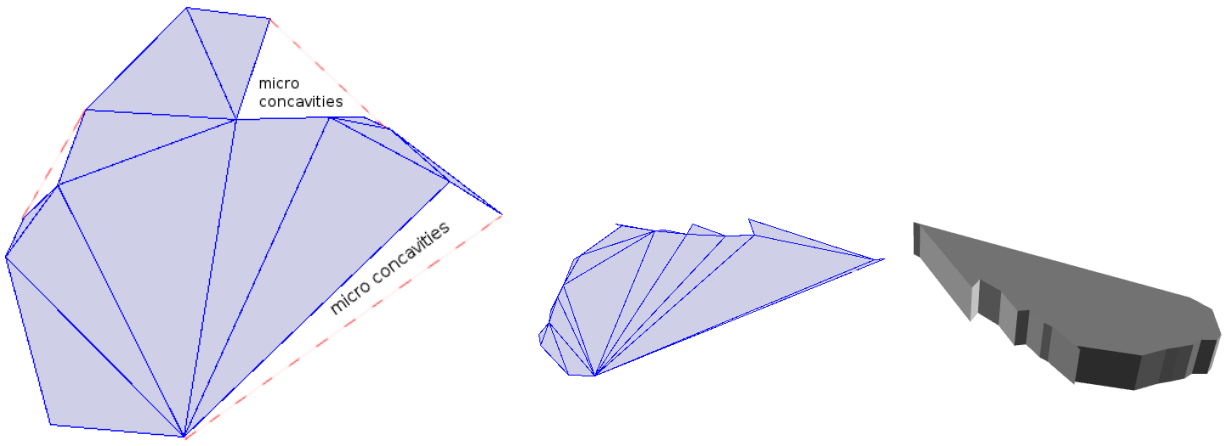
Figure 6.3. A demo algorithm using *Triangle* software on a fictive roof segment assuming that outer edges are already known. First we have tried a common Delaunay triangulation on its convex hull (b). Next we generate a polygon from its side edges as described in Section 6.1.1 (c). We produced a PLSG triangulation on the polygon with *Triangle* retrieving a rough polygon mesh (d). After eliminating triangles with no neighbour triangles (triangles connecting just by one corner) we obtain a simplified polygon mesh (e) with regular outer sides (f).



(a) detected edges (b) generated polygon (c) triangulation with Triangle

```
function generate_polygon(in edgelist, out polygon)

    -- chosing one of the two endpoints (eg. 'a') of the first edge
    -- this will be the reference point at the first iteration
    reference = edgelist.first_edge.endpoint_a

    while (reference exists)
        -- finding the closest free endpoint to the actual reference point
        new_endpoint = closest free endpoint to reference

        -- appending the previously selected edge represented by its endpoints
        -- having the assurance it exists
        polygon.append( Edge(reference, reference.pair) )

        -- changing the reference to the other endpoint of the newly selected edge
        reference = new_endpoint.pair
    end
end
```

(d) A brief pseudo code of polygon generation from the detected edge segments.



(e) The generated shape includes micro concavity, that will result in a fragmented model.

(f) An illustration of the weakness of the PSLG method on a real roof segment. The generated mesh has very rough horizontal walls that can not be accepted.

Figure 6.4. Results of polygon generation and PSLG triangulation on real problems

6.1.1 Experiments with the PSLG method

The needed polygon is to be produced from the topologically sorted boundary lines of the respective roof segment.

Let us call *free* those edges that do not take part in the polygon yet. Consequently an endpoint is *free* if it belongs to a free edge. Now, we choose an arbitrary edge to be the first side of the polygon. Next, in each iteration we find the closest free endpoint to last edge's endpoint in the polygon. The closest endpoint of the newly selected edge is linked to the endpoint of the last edge in the polygon. In the next iteration we will produce the same but now the reference point will be the non-linked endpoint of the newly selected edge. The iteration is stopped when no free edges are found. In Figure 6.4d we can see a brief pseudo code of the algorithm.

In fact the roof segments roughness and the irregular arrangement of the boundary lines lead to fragmented polygons having several micro-concavities on the boundaries, thus the observed results do not proved to be acceptable.

6.2 Concave problem - stochastic approach

As we have seen in Figure 6.4, the previously generated shapes often show some degree of concavity, therefore they are very fragmented resulting unappealing rough building blocks. The convex hull has a little smoothing effect on the 2D meshes concealing micro-concavities and boundary irregularities.

In this section we aim to design an algorithm that prevents convex triangulation on relatively large concave areas (bays) but preserves smooth edges on other parts of the mesh filling in the micro-concavities. Let us denote the set of all triangles by S.

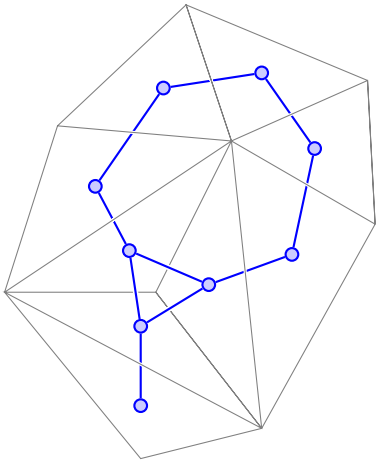


Figure 6.5. 3D triangle mesh and its corresponding undirected (triangulation) graph

According to Gansner, Hu and Kobourov et al. [33] a triangle mesh is defined by the included triangles (S) and the neighborhood connections ($\mathcal{N}$) between them, hence it can be modeled as an undirected graph (Figure 6.5) where each triangle is considered as a separate vertex ($s \in S$), and each neighborhood connection as an undirected link ($\{s,r\} \in \mathcal{N}$) between two vertices ($s,r \in S$) corresponding to the neighboring triangles. Furthermore, some of the triangles in the mesh need to be deleted because they are lying on the concave parts of the roof segment. As a consequence, we have to assign a random variable (Ω_s) to each vertex that marks the fact that the corresponding triangle needs to be eliminated or not. After classification, we call a given triangle as *relevant triangle*, if it should be kept in the final mesh. These Ω_s variables are defined on the set of vertices (S), therefore they form a $\Omega = (\Omega_s)_{s \in S}$ random field with respect to $\mathcal{N}$.

Markov property

In our case *planar graph* models can also be used, since the considered meshes are open[2] (i.e. they have at least three triangles having neighbors fewer then three), and do not contain any wholes. *Planar graphs*[3] can be drawn on the plane, so that their edges intersect only at their endpoints. It is convenient to use them, since they satisfy the below constraints[4]:

- we have an upper limit for the number of edges: $e \leq 3v - 6$, where e is the number of edges and v is the number of vertices.
- the maximum number of vertices of a fully connected (complete[5]) sub-graph is 4.

With reference to Gansner, Hu and Kobourov et al. [33] (Lemma 1. on pg. 5.) we cannot draw on the plane 4 triangles which are all connected to each other, therefore the maximum number of vertices of a complete sub-graph is 3. Accordingly, the vertices of the graph are usually connected with a reduced number of other vertices especially in their close proximity (Markov property). Furthermore, we presume that every triangle's label is conditionally independent of any other non-adjacent node's label, given the labels of all neighboring triangles. With regard to Stan Z. Li et al. [20], Ω is said to be a Markov Random Field on S with respect to $\mathcal{N}$.

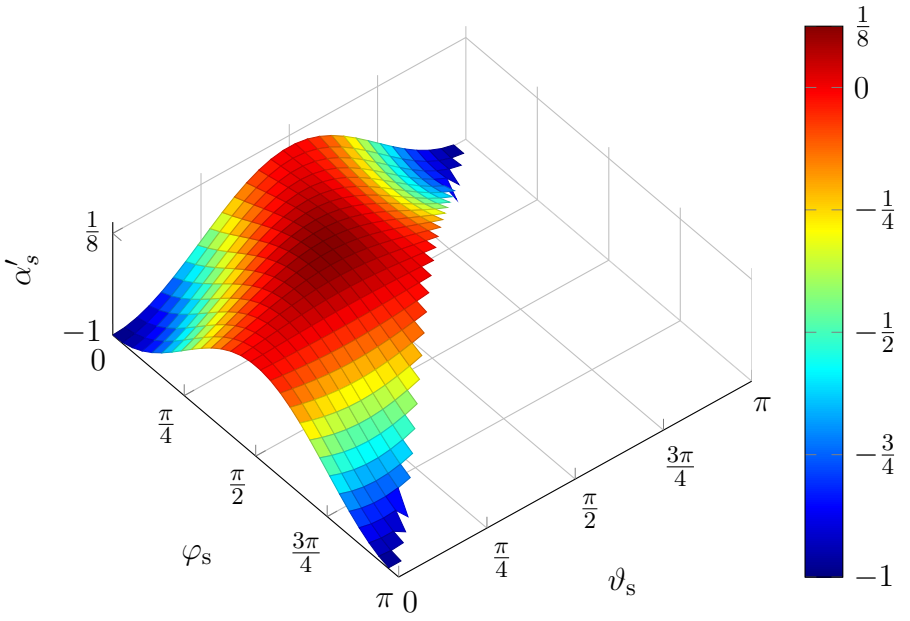


Figure 6.6. Demonstration of the angle cost $(\alpha'_s) = \cos(\varphi_s)\cos(\vartheta_s)\cos(\pi - \varphi_s - \vartheta_s)$. In the region $\mathcal{R} = \left\{(\varphi, \vartheta) \;\middle|\; \varphi, \vartheta > 0;\ \varphi + \vartheta < \pi\right\}$, α'_s has its maximum (0.125) if $\varphi = \vartheta = \frac{\pi}{3}$, and its minimums (-1) if one of its three angles tends to be zero.

[2] they are open 2-manifolds (Smith et al. [34] on pg. 14.)

[3] see definition given by Balakrishnan et al. [35] (Definition 8.2.1 on pg. 175.)

[4] see theorems and their consequences formulated by Balakrishnan et al. [35] in Section 8.3 *Euler Formula and Its Consequences*

[5] see definition given by Balakrishnan et al. [35] (Definition 1.2.11)

Prior and observation model

As for the ***prior model***, we used the energy function presented in Section 2.1.2. Regarding to equation (2.4), the *smoothness energy* is defined as follows:

$$E_s = \sum_{\{s,r\}\in\mathcal{N}} V(\omega_s, \omega_r)$$

where V implements again a smoothing constraints, using the Kronecker delta: $V(\omega_s, \omega_r) = \delta(\omega_s = \omega_r)$. In case of a particular $s \in S$ triangle, our ***observation model*** uses the following descriptors:

n_s : number of points projected into s
A_s : area of s
φ_s, ϑ_s : two arbitrary angles of s

Using these measures we will generate a single x_s value for each triangle s, so that x_s will be approximately proportional with the likelihood of the fact that s constitutes a relevant element in the concave triangulation, which should not be deleted from the final mesh. As for the first feature, we calculate the density of the projected points in each triangle ($\frac{n_s}{A_s}$), and we divide the calculated value by a

$$\mathcal{K} = \max_{s_i \in S} \frac{n_{s_i}}{A_{s_i}}$$

normalization coefficient, so that we obtain a density feature in the interval $[0, 1]$. Let us introduce the following notation for this descriptor (ρ stands for density):

$$\rho_s = \frac{1}{\mathcal{K}} \cdot \frac{n_s}{A_s} \ \in \ [0, 1] \tag{6.1}$$

Next, we use our observation that bays (i.e. internal regions of the mesh which should be likely eliminated) contain mainly acute-angled triangles, while micro concavities on the boundaries of the open mesh consist of long and thin obtuse triangles. As a consequence, we introduced a so-called *angle cost* that measures how much a given triangle is obtuse. Let us define *angle cost* as the product of each angle's cosine values in the triangle.

$$\alpha'_s = \cos(\varphi_s)\cos(\vartheta_s)\cos(\pi - \varphi_s - \vartheta_s) \qquad \in \ [-1, 0.125]$$
$$\alpha_s = \Big(\cos(\varphi_s)\cos(\vartheta_s)\cos(\pi - \varphi_s - \vartheta_s) + 1\Big) \cdot \frac{1}{1.125} \qquad \in \ [0, 1]$$

As illustrated on the 3D plot in Figure 6.6, the angle cost gives its maximum value (0.125 without scaling), if the triangle is equilateral ($\varphi = \vartheta = \pi/3$). Otherwise, the more a triangle is acute, the more its angle cost tents to -1.

Finally, a joint fitness value, i.e. a pseudo probability is defined as the product of ρ and $(1 - \alpha)$, which indicates us whether, a given triangle is a relevant convex part of the mesh. These $x_s = \rho_s \cdot (1 - \alpha_s)$ values will form our ***observed features***.

However, during our experiments we perceived that excluding triangles just by their low x_s values using a given hard threshold can cause several false positive/negative triangles. In

other words, the designed feature (x_s) is not enough in itself to decide whether a triangle belongs to the concave parts of the mesh or not. To overcome this limitation, we started to compare each triangle's class label with labels in its neighborhood, hence we took the advantage of the *prior model*. Just for illustration (Figure 6.7), let us color relevant triangles red and triangles able to be skipped gray. This color can also be interpreted as a label marking that the corresponding triangle is *relevant* or *removable*. If two or three neighboring triangles are gray the actual triangle is likely to be gray too, especially when the nearby triangles have larger area then the actual one.

For the $d_s(\omega_s) = d(x_s \mid \omega_s)$ *data cost* I have chosen the following functions:

$$d(x_s \mid \Omega_s = \text{relevant}) = f(1 - x_s)$$
$$d(x_s \mid \Omega_s = \text{removable}) = f(x_s) \qquad \forall\ s \in S$$

where $f(x)$ is the sigmoid function, operating as a soft threshold:

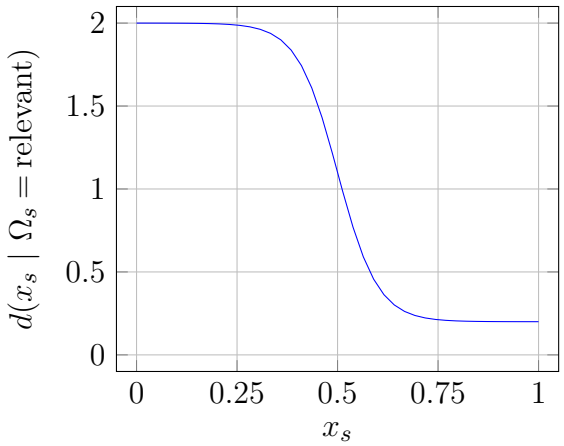
$$f(x) = \frac{1}{1 + e^{-n(x - x_0)}} \cdot (b - a) + a, \qquad f : [0,1] \longrightarrow [a,b]$$

with gradient $n = 20$, shift $x_0 = 0.5$, offset and scale $(a,b) = (0.2,2)$. Thereafter the *data model* of the MRF has the following form:

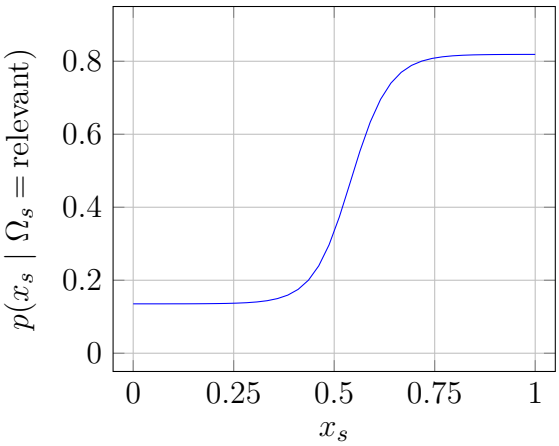
$$p(x_s \mid \omega_s) = e^{-d(x_s \mid \omega_s)}$$

Note that the above quantities define pseudo probabilities, since they are unnormalized.

Data cost function if Ω_s is *relevant*



Corresponding pdf if Ω_s is *relevant*



Using our defined *prior* and *observation model*, we can determine the ***posterior*** likelihood $P(\omega \mid x)$ of every possible global labeling over the triangle-graph, and we have no other tasks but choosing the most probable global labeling that will point out the desired (*relevant*) triangles. According to equation (2.7) in Section 2.1.2, we have to optimize the following energy function using graph cuts based optimization technique developed by Olga Veksler, using the libraries provided by Yuri Boykov and Vladimir Kolmogorov [25–28]:

$$\omega_{opt} = \arg\min_{\omega \in \Gamma} \left(\sum_{s \in S} d(x_s \mid \omega_s) + \lambda \sum_{\{s,r\} \in \mathcal{N}_s} \delta(\omega_s, \omega_r) \right) \tag{6.2}$$

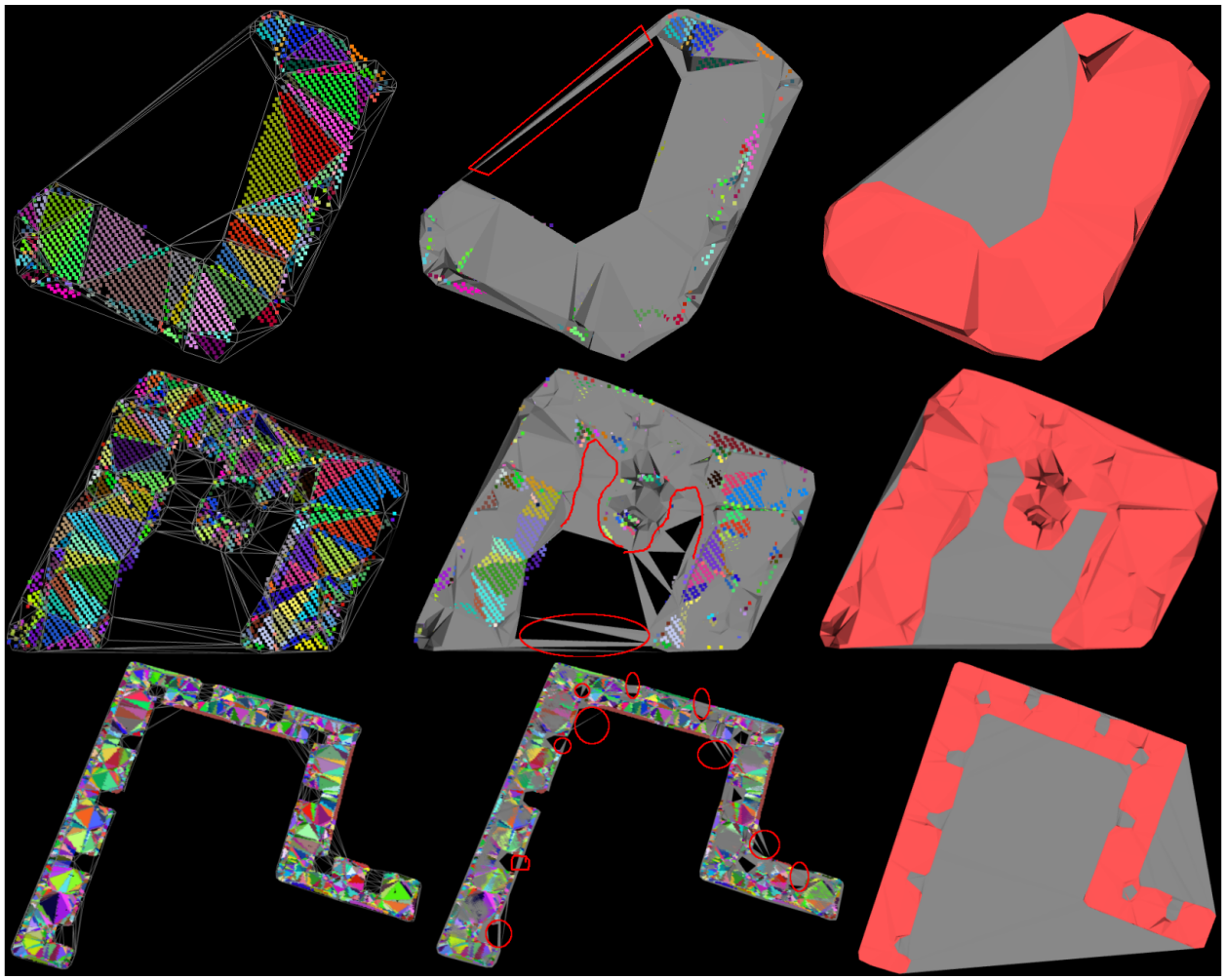


Figure 6.7. Removing triangles that do not belong to the concave hull. The first two columns demonstrate a hard threshold of the x_s feature, without the MRF smoothing constraint: we can observe several false triangles in the resulting meshes (2nd column). The 3rd column shows the optimal label mask where maximum probability is met. In the first column it can also be observed through coloring how the points are associated to the corresponding triangles.

6.2.1 Calculating the ρ_s density feature

As we have seen by equation (6.1), the ρ_s density factor is calculated using the n_s and A_s descriptors, from which A_s is calculated in $\mathcal{O}(1)$ fashion by the cross product of two side vectors ($\vec{a}$, $\vec{b}$) of triangle s:

$$A_s = \frac{\|\vec{a} \times \vec{b}\|}{2}$$

On the other hand calculating n_s consumes exceedingly much time especially if we do not use any space-partitioning techniques, because we have to loop over the roof segment's points checking if each point falls into the s triangle.

This procedure's complexity for a single roof segment is $\mathcal{O}(N \cdot K)$, where N and K are the numbers of points respectively triangles. For example, in case of the relatively large[6] roof component in Figure 6.7 (at the bottom) the processing time is approximately 2 minutes.

We have already gained experiences, how much can valuable time can be saved by using only 2D images instead of 3D cloud processing. Thus we use a planimetric projection again, but now we produce two images with the same size: a colored and a binary image. Before projection we rotate the cloud to make it parallel to the xy plane and each pixel that a point has been fallen onto is colored white. As a result we obtain a leaky white shape with black background, as illustrated in the 1st image of Figure 6.8. The holes can be filled by some number of binary dilation (2nd image of Figure 6.8).

In the following we generate a mask (a colored image) by projecting each triangle onto to mask using different colors for every single triangle. Not to mention, that the polygon mesh is already in 2D, since the projection is inevitable during the triangulation.

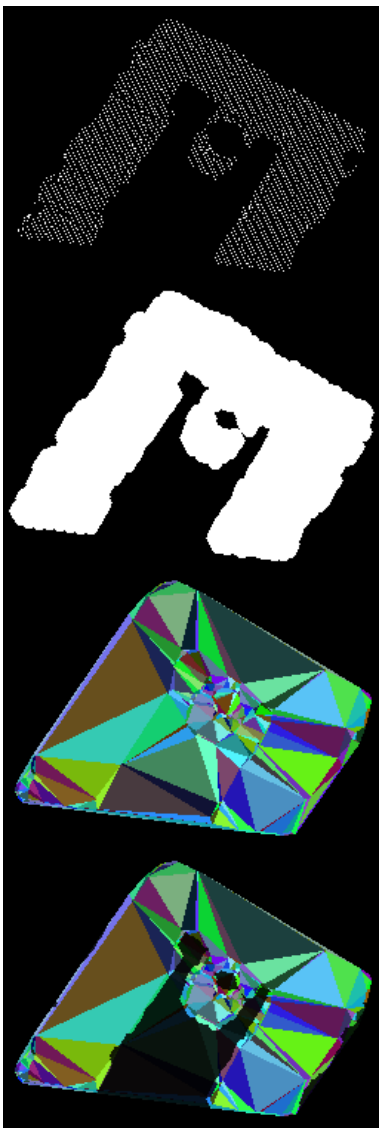


Figure 6.8. Projected binary image (1st), dilated binary image (2nd), triangle mask (3rd), triangle mask covered by the dilated binary image (4th)

After the two images have been created, we compare them by counting how many white pixels fall onto the colored mask in each triangle (3rd image of Figure 6.8. Then the density measure, that we introduced in the previous section, is going to be adjusted by the proportion n'_s/A'_s, where A'_s is the area of s triangle's colored projection onto the mask, and n'_s is the number of white points on the binary image located within this s triangle's area. To put it another way, n'_s is the number of luminous pixels in s triangle of the last image of Figure 6.8, which have been produced by covering the colored triangle mask with the dilated image of the the concave roof segment.

[6] having 35778 points

At the same time, statistics shows that very thin triangles with small areas appear quite frequent and their projection's area is zero due to the limited resolution of the triangle mask image (i.e. the global labeling). To avoid division by zero we use a kind of correction:

$$\rho_s = \frac{n'_s + k}{A'_s + k + 1}$$

For practical reasons we allowed k to be 9, which means that we give quite a good chance for small triangles to take part in the final concave mesh, but they can easily be removed by their larger neighbors.

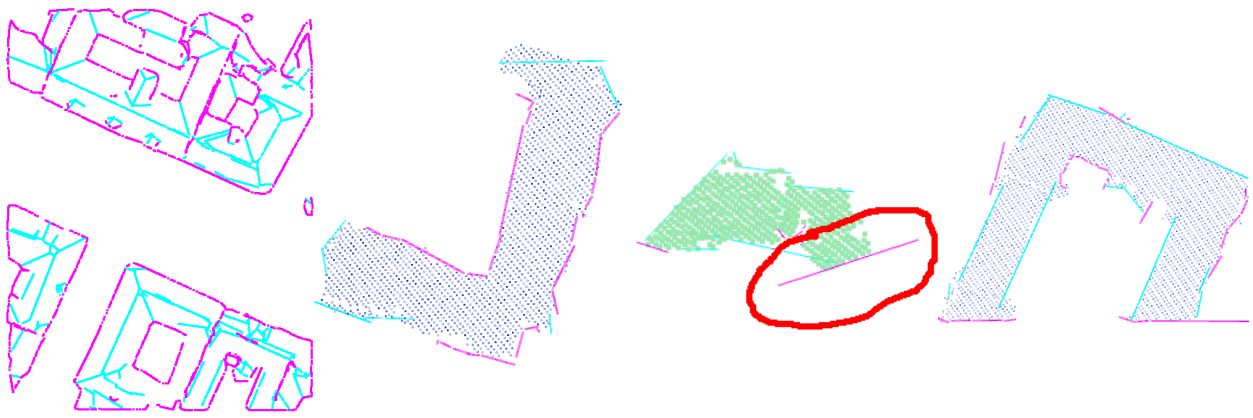
As for the processing time, it takes less than 2 seconds for reconstructing the roof segment of Figure 6.7 (at the bottom). Besides its speed, this method has another advantage, namely, the fact that it handles the problem of boundary micro concavities by using a few additional dilation. Dilating three-four times the binary image, triangles on the concave parts will be covered in some degree by the white object of the dilated image (i.e. the image of the roof segment).

7. A detailed assessment of the algorithm

The roof segmentation algorithm produces an adequate result, the great majority of planar roof sections are distinguished, and the neighboring roof sections are touching. We have experienced a reliable performance of the initial classification and the inner edge detection steps.

One part of the observed errors has been produced by the outer edge detection step, because the procedure has been executed without knowing anything about roof components. Consequently, false common edges, which belongs to different neighboring roof segments, have also been detected in some cases (Figure 7.1c, 7.1d).

On the other hand, the algorithm tries to generate a detailed model, therefore rough lines may be fragmented into very short pieces, as illustrated in Figure 7.1b. Furthermore, no simplification or generalization is made before performing the mesh generation. As presented by Zhou et al. [13] global regularities could be found and an initial simplification could be performed for the sake of an improved flexibility. However as Lafarge et al. [5] states, that these presumptions proposed by Zhou would fit "Manhattan-world" environments and "well-organized" urban landscapes, but they are less efficient for towns with old architectural buildings, historical towers like in several districts of Budapest.



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

Figure 7.1. Demonstration of the edge detection step, with also featuring the limitation of the approach (image c). The turquoise segments are inner edges, the pink ones are outer edges.

Processing time

The main CPU consumer module of the system is the roof segmentation step, where a moving least squares (MLS) algorithm is executed followed by a complex point cloud filtering process in the three-dimensional space. Nevertheless, the system is offline, our aim was to improve a reasonable processing time which has been achieved. In case of data belonging to a measurement of $0.5km^2$ area (corresponding to a cloud having 8 million points) takes in one and a half hour in a standard desktop computer.

7.1 Strength of the system

First, the algorithm was designed using a training cloud presented in Figure 1.1 containing approximately 100 thousands of points. Then we tested the system on different clouds presenting several types of buildings. The core algorithm did not require any modifications but the former application framework needed some optimizations, generalizations and corrections in order to be able to process the algorithm on larger clouds.

After minor improvements of my basic configuration, the software generated quite impressive models, even for old architectural constructions like the central "K" building of the Budapest University of Technology and Economics or the Grate Marked Hall (Vásárcsarnok) alongside Vámház boulevard in Budapest. Notice, that both of them have two square based pyramid shaped towers on their front side. To our firm opinion, the obtained reconstruction results are quite acceptable considering that the system was initially designed for planar roof shapes. On the upper images in Figure 7.2 you can see a larger city area with old civil apartment houses, where different irregular constructions appear, like the garage in the red frame in Figure 7.2 surrounded by towering civil houses. These asymmetries do not confuse the system, moreover the concerning building shapes are precisely reconstructed.

Figure 7.3b demonstrates a typical "Manhattan-world" environment proving that my implementation performs well on these types of urban scenes too.

As a further development, I have optimized the system, making it appropriate for processing even larger clouds as illustrated in Figure 7.3a. Here we have performed an initial fragmentation using *libLAS*[1]'s *las2las* software, which splits up the the original dataset into multiple clouds having at most 2 million points each. The resulting large fragments are separated into connected building blocks, and our algorithm guarantees a reasonable model for each block.

[1] `http://www.liblas.org/`

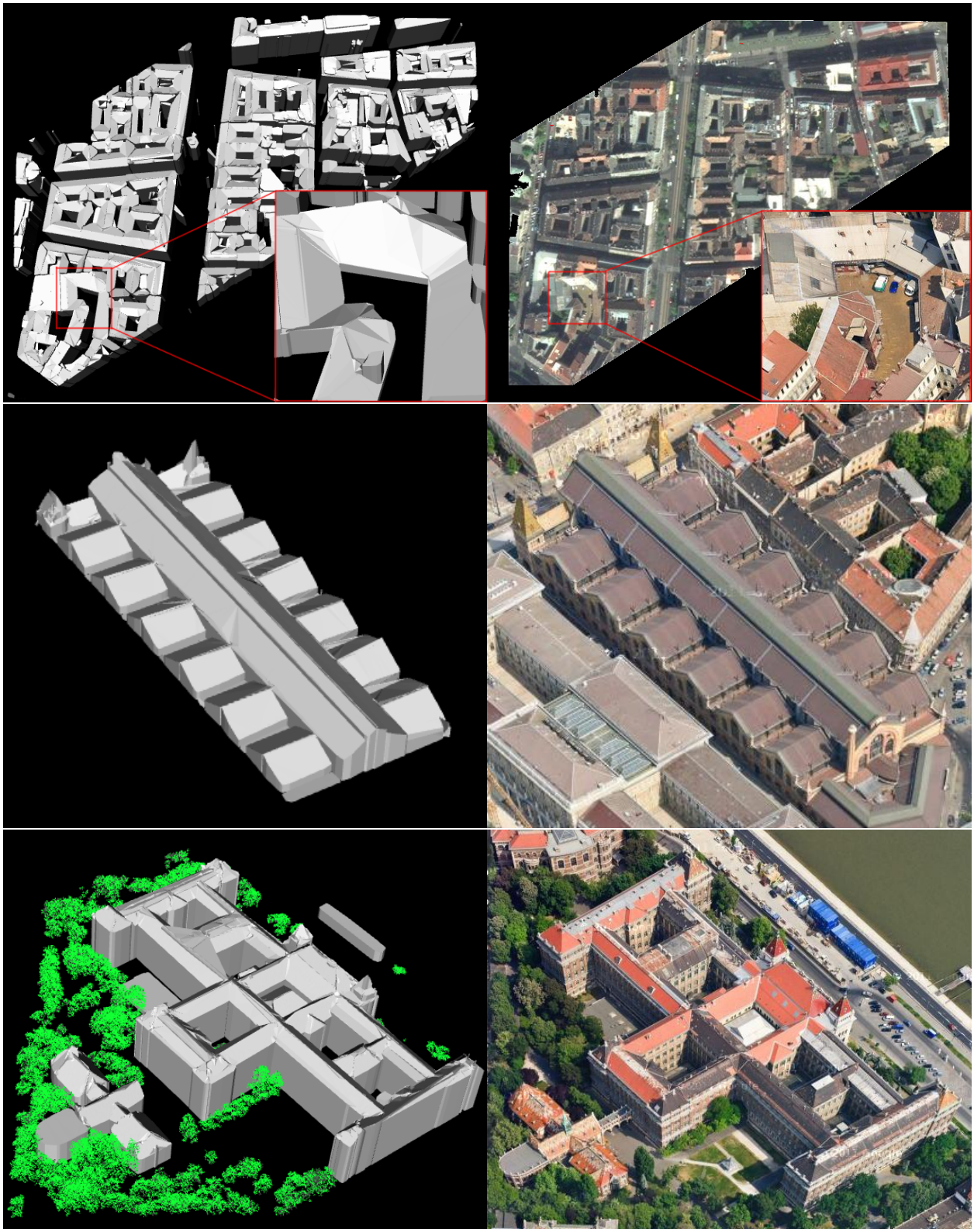
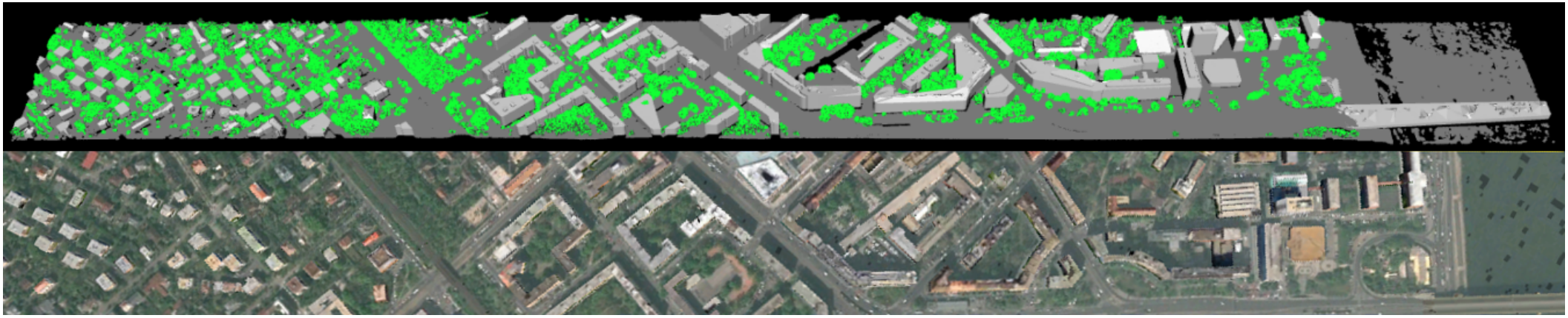
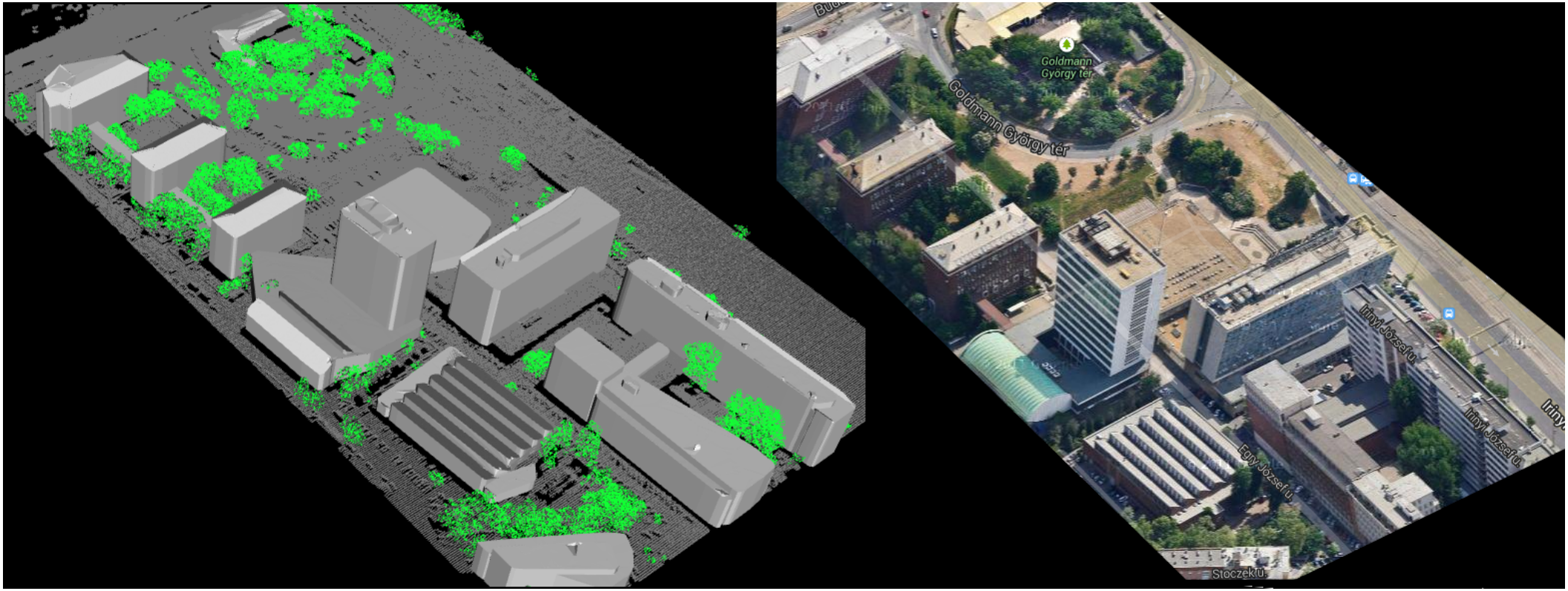


Figure 7.2. Our polygon reconstruction results (left), and the reference aerial photos (right) of various landmarks of Budapest. Civil apartment houses - Budapest's site between Mária St., Nap St., Futó St. and Baross St. (top), Vásárcsarnok Market, Vámház körút (middle), Budapest University of Technology and Economics (BUTA), K building (bottom)



(a) Large city area on the South part of Buda near Móricz Zsigmond square. Left side of the model presents rural scenes with family houses and smaller flat buildings. Large concave blocks appear in the middle of the image.



(b) Towering flat buildings near Petőfi bridge, Buda abutment at Goldmann György square. These constructions resemble a “Manhattan-world” environment.

Figure 7.3

8. Framework for displaying and processing data

In this chapter I intend to give a brief description about the used software libraries and the implemented framework.

8.1 Development tools

The core functions are written in the C++ programming language on Linux platform which are compiled into several dynamic libraries according to the specific modules. These subroutines are called from external scripts written in Python2.7 programming language, where all necessary parameters can be set without the need of recompiling the whole software again. There is a bash script which executes the initial fragmentation using the *las2las* software. Finally the same bash script performs the reconstruction algorithm for each fragment using the Python scripts. Demo visualizer applications for tests and demonstrations were developed using Qt4.7 library.

8.2 Point Cloud Library

The Point Cloud Library[1] (PCL) is a standalone, large scale, open project for 2D/3D image and point cloud processing. I have used PCL's type representations (point types, point cloud) and several data structures, algorithms (kd-tree structure, filtering, etc...).

8.3 Computational Geometry Algorithms Library

The Computational Geometry Algorithms Library[2] (CGAL) provide easy access to efficient and reliable geometric algorithms in the form of a C++ library. In my project I have used CGAL's 3D object representations (vector, point, plane, transform matrix, etc...), 3D geometrical and linear algebra operations (plane intersection, Euclidean distances, matrix transformations, etc...) and several triangulations (2D simple and constrained triangulations, Delaunay triangulations).

[1] et al. [37] (`http://www.pointclouds.org/`)
[2] et al. [38] `http://www.cgal.org/`

8.4 Simplified Wrapper and Interface Generator

The Simplified Wrapper and Interface Generator[3] (SWIG) is a software development tool (SDK) that connects programs written in C and C++ with a variety of high-level programming languages (eg. Python). I have used this SDK to generate Python scripts, which are able to call C++ functions compiled into shared libraries.

8.5 Implemented framework

The main functions are implemented in C++ and are subdivided into several modules, which are listed as follows:

basic synthesis
: initial point cloud classification, connected roof cloud extraction (Chapter 3)

roof segmentation
: estimating surface normals, filtering normals, clustering, roof point labeling and segmentation (Chapter 4)

outer edge detection
: an assembly which detects outer outer boundary points (Figure 5.1)

segment fitting
: finds the 3D boundary lines which fit the edge cloud, the last step of outer line detection (Figure 5.1 - last figure)

reconstruction
: a wrapper classes which can handle inner edge detection, skeleton generation (Chapter 5), triangulation using CGAL, calculating x_s feature for all triangle, etc... (Section 6.2.1)

MRF
: a wrapper classes for the MRF APIs developed by Olga Veksler, using the libraries provided by Yuri Boykov and Vladimir Kolmogorov (Chapter 6)

demonstration module
: is a complex Qt based application, in which I have visualized the intermediate results (Figures 1.1, 5.1, 6.1, 6.3, 6.7, 7.1)

PCL types
: an auxiliary module, in which I have developed a wrapper class around PCL's *PointCloud* using C macros. This wrapper gave us the possibility to manage *PointCloud* objects in Python scripts in a straightforward way (loading from / write to external file, load cloud to a PCL's kd-tree, conversion between clouds and Python lists, color cloud points, conversion between PCL's different *PointCloud* representations). Furthermore we can pass cloud objects between Python and C++ functions.

[3] `http://www.swig.org/`

9. Future plans and possible improvements

In the future, we aim to use the developed algorithms in the ongoing DUSIREF project of the Distributed Events Analysis Research Laboratory of MTA SZTAKI, which project is funded by the European Space Agency. Here the synthesized city model will provide a 3D visualization environment, where the results of different terrestrial or aerial LiDAR based computer vision and machine recognition procedures can be displayed.

Furthermore, the obtained three-dimensional models will be compared with optical images taken from the space in different times, analysing the possibilities of adaptive texturing and change detection.

I also plan to improve the proposed algorithms at certain points. In order to achieve a more precise model, the outer segment fitting step should be reconsidered because that is the weakest link in the whole algorithm even though it is relatively fast. I plan to implement a least mean square based line fitting in the knowledge of the tangent vector field of the edge cloud in each edge point. For example a RanSaC can be applied that will fit a line to the longest edge segment. Then points in the proximity of the previously defined line will be removed from the edge cloud and we do it again until the cloud becomes empty. An important question for the future is to determine the maximal reachable processing speed.

Further possibilities of development can be extracting topologically complex ground surfaces with significant slopes using stochastic segmentation approaches and modelling them efficiently by sparse triangle meshes. These types of sites appear frequently even in Budapest, for example we can mention the Gellért Hill, or scenes around Buda Castle.

10. Conclusion

My work's primary objective was to design an automatic and robust method to process aerial LiDAR data and produce three-dimensional geometric models from them. Indeed, a long way have had to be taken during the last ten months to pursue our objective, and many obstacles appeared, especially in feature line detection and mesh generation steps.

My main contributions consist of the proposed robust roof segmentation, the edge detection algorithm, and the concave mesh generation procedure based on a probabilistic graphical model. Further improvements have been made in point cloud classification, in which we have extracted vegetation and buildings. I optimized the method by splitting up the cloud (Section 3.1), using image processing algorithms instead of 3D point cloud processing, and introduced several automatic steps (Section 8.1) which give us the possibility to test our algorithm on large point clouds.

As the results illustrate, the algorithm can be applied for a wide range of building types even though it solely estimates the geometry of objects by several planar elements (polygons). We have reconstructed city sites featuring urban civil apartment houses (Figures 1.1 and 7.2 - city site), buildings with complex architectural roof models (Figure 7.2 - Market Hall and BUTE K-building), large concave blocks of flat (Figure 6.2 and 7.3b), rural (Figure 7.3a), and "Manhattan-world" environments (Figure 7.3b).

11. Acknowledgement

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to my supervisor, Dr. Benedek Csaba, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project especially in writing this report.

Furthermore, I say many thanks to the head of the Distributed Events Analysis Research Laboratory (DEVA), Dr. Szirányi Tamás, for his valuable and constructive suggestions. My special thanks are extended to the staff of DEVA of MTA SZTAKI, especially to Börcs Attila, who assisted me in many cases.

Last but not least, I would also like to acknowledge with much appreciation the crucial role of Infoterra Astrium GEO-Information Services ©, who gave us the permission to test our system on their aerial LiDAR records of Budapest.

This work was supported by the Government of Hungary through a European Space Agency (ESA) Contract under the Plan for European Cooperating States (PECS), and by the Hungarian Research Fund (OTKA #101598).

Bibliography

[1] Attila Börcs and Csaba Horváth. Városi környezet automatikus analízise és rekonstrukciója légi LiDAR mérések alapján, TDK dolgozat, Pázmány Péter Katolikus Egyetem Információs Technológiai Kar. 2011.

[2] Attila Börcs and Csaba Benedek. Urban traffic monitoring from aerial LIDAR data with a two-level marked point process model. In *International Conference on Pattern Recognition*, Tsukuba City, Japan, 2012.

[3] Attila Börcs and Csaba Benedek. A marked point process model for vehicle detection in aerial LIDAR point clouds. In *ISPRS Annals of Photogrammetry, Remote Sensing and the Spatial Information Sciences (Proceedings ISPRS Congress)*, Melbourne, Australia, 2012.

[4] Florent Lafarge and Clément Mallet. Creating large-scale city models from 3d-point clouds: A robust approach with hybrid representation. *International Journal of Computer Vision*, 99(1):69–85, August 2012.

[5] Florent Lafarge and Clément Mallet. Building large urban environments from unstructured point data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1068–1075, Barcelona, Spain, 2011.

[6] F. Lafarge, X. Descombes, J. Zerubia, and M. Pierrot-Deseilligny. Structural approach for building reconstruction from a single dsm. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(1):135–147, 2010.

[7] Hai Huang, Claus Brenner, and Monika Sester. 3d building roof reconstruction from point clouds via generative models. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 16–24, New York, NY, USA, 2011. ACM.

[8] Hai Huang, Claus Brenner, and Monika Sester. A generative statistical approach to automatic 3d building roof reconstruction from laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 79(0):29–43, May 2013.

[9] Qian-Yi Zhou and Ulrich Neumann. Complete residential urban area reconstruction from dense aerial LIDAR point clouds. *Graphical Models*, 75(3):118–125, 2013.

[10] Qian-Yi Zhou and Ulrich Neumann. Modeling residential urban areas from dense aerial LIDAR point clouds. In *Proceedings of the First international conference on Computational Visual Media*, CVM'12, pages 91–98, Berlin, Heidelberg, 2012. Springer-Verlag.

[11] Qian-Yi Zhou and Ulrich Neumann. 2.5d building modeling by discovering global regularities. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 326–333, 2012.

[12] Qian-Yi Zhou and Ulrich Neumann. 2.5d building modeling with topology control. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2489–2496, 2011.

[13] Qian-Yi Zhou. *3D urban modeling from city-scale aerial LIDAR data*. Phd thesis, Faculty of the Graduade School University of Southern California, 2012.

[14] Vivek Verma, Rakesh Kumar, and Stephen Hsu. 3d building detection and modeling from aerial LIDAR data. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2 of *CVPR '06*, pages 2213–2220, Washington, DC, USA, 2006. IEEE Computer Society.

[15] Abdullatif Alharthy and James Bethel. Heuristic filtering and 3d feature extraction from LIDAR data. In *In ISPRS Commission III, Symposium 2002 September 9 - 13*, pages 23–28, September 2002.

[16] William K. Pratt. *Digital Image Processing: PIKS Inside*. John Wiley & Sons, Inc., New York, NY, USA, 3rd edition, 2001.

[17] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, 2 edition, 2001.

[18] G. Bradski and A. Kaehler. *Learning OpenCV: computer vision with the OpenCV library*. O'Reilly Series. O'Reilly, 2008.

[19] Guillaume Lavoué and Christian Wolf. Markov random fields for improving 3d mesh analysis and segmentation. In *Eurographics 2008 Workshop on 3D Object Retrieval*, April 2008.

[20] Stan Z. Li. *Markov Random Field Modeling in Image Analysis*. Springer Publishing Company, Incorporated, 3rd edition, 2009.

[21] Ross Kindermann and J. Laurie Snell. *Markov Random Fields and Their Applications*. American Mathematical Society (AMS), 1980.

[22] John Moussouris. Gibbs and Markov random systems with constraints. *Journal of statistical physics*, 10(1):11–33, 1974.

[23] H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications*, volume 104 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London, 2005.

[24] J. M. Hammersley and P. Clifford. Markov field on finite graphs and lattices. 1971.

[25] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Aseem Agarwala, and Carsten Rother. A comparative study of energy minimization methods for markov random fields. In *In ECCV*, pages 16–29, 2006.

[26] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, November 2001.

[27] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:65–81, 2004.

[28] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, sep 2004.

[29] Rui Hu, Stuart James, Tinghuai Wang, and John P. Collomosse. Markov random fields for sketch based video retrieval. In *ICMR*, pages 279–286, 2013.

[30] Stephan Scholze, Theo Moons, and Luc J. Van Gool. A probabilistic approach to building roof reconstruction using semantic labelling. In *Proceedings of the 24th DAGM Symposium on Pattern Recognition*, pages 257–264, London, UK, UK, 2002. Springer-Verlag.

[31] Jean Gallier. Notes on Convex Sets, Polytopes, Polyhedra Combinatorial Topology, Voronoi Diagrams and Delaunay Triangulations. Rapport de recherche RR-6379, INRIA, 2007.

[32] P.L. George and H. Borouchaki. *Delaunay Triangulation and Meshing: Application to Finite Elements*. Butterworth-Heinemann, 1998.

[33] Emden R. Gansner, Yifan Hu, and Stephen G. Kobourov. On touching triangle graphs. *CoRR*, abs/1001.2862, 2010.

[34] Colin Smith. *On vertex-vertex systems and their use in geometric and biological modelling*. PhD thesis, Calgary, Alta., Canada, Canada, 2006. AAINR19574.

[35] R. Balakrishnan and K. Ranganathan. *A Textbook of Graph Theory*. Universitext - Springer-Verlag. Springer, 2012.

[36] Hervé Abdi and Lynne J. Williams. Principal component analysis, 2010.

[37] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *International Conference on Robotics and Automation*, Shanghai, China, 2011.

[38] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.