EPITA–École Pour l'Informatique et les Techniques Avancées
CSI–Calcul Scientifique et Image
LRDE–Laboratoire de Recherche et Développement de l'EPITA

Improving point cloud support of *ContextCapture*^{TM} : Scan Finder, Point Cloud Visibility and Point Cloud Compression

Alexandre Gbaguidi Aïsse

Supervised by Cyril Novel

Submitted in part fulfilment of the requirements for the degree of
Software Engineer of EPITA, Paris, August 2018.

Abstract

Text of the Abstract.

Acknowledgements

I would like to express (whatever feelings I have) to:

- My supervisor
- My second supervisor
- Other researchers
- My family and friends

Dedication

Dedication here.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The need of 3D realistic models is increasingly present in several fields such as architecture, digital simulation or civil and structural engineering. One way to obtain a 3D model might be to build it by hand using specialized modeling softwares. In this case, the realistic aspect of the model could be doubtful. A more reliable way is to use *photogrammetry* or *surface reconstruction* or both at the same time. *ContextCapture*^{TM} is a reality modeling software that can produce highly detailed 3D models. It creates models of all types or scales from simple photographs of a scene to point clouds or both of them thanks to its hybrid processing. The usage of point clouds gained wide popularity because of the emergence of devices such as optical laser-based range scanners, structured light scanners, LiDAR scanners, Microsoft Kinect, etc. Actually, it is only in the past two years that *ContextCapture*^{TM} has started to support point clouds and the common goal between each part of this internship is to improve *ContextCapture*^{TM} point cloud support.

The general problem being solved by *surface reconstruction* is: given a point cloud P assuming to lie near an unknown shape S, construct a digital representation D approximating S. In order to reconstruct point clouds acquired from static[1] LiDAR scanners, possibly multi-scan[2], *ContextCapture*^{TM} needs to know the position of each scanner in the scene, on the one hand, and the attribution of each point to a scanner on the other. The scanners location information is not always present in point clouds metadatas, for instance LAS file format does not provide it. Moreover, some users of *ContextCapture*^{TM} sometimes lost this information due to a prior export with no metadata. Detecting the positions of multiple scanners exclusively from a point cloud is a subject not identified as inter-

[1] As opposed to mobile LiDAR scanners.
[2] Point cloud made of several laser scans.

esting by academics, and thus not tackled since this information is almost always available from the outset. The same holds true on the industry side, it is only recently that scanner position information is relevant for a few applications like *ContextCapture*^{TM} . This is the main contribution presented in this internship report: a method able to detect automatically multiple scanners in a single point cloud without prior knowledge of the scene.

Knowing scanners position is one step toward supporting LAS point cloud format. *ContextCapture*^{TM} still needs to know for each point which scanner sees it best[3]. This enters into the realm of visibility of point clouds. One way to retrieve visibility of point clouds is to reconstruct the surface and use the underlying mesh to compute visibility. But to reconstruct the surface we need to orient the normals; a chicken-and-egg problem. After some literature review on the subject, we did not find accurate method for LiDAR point clouds. Also, most papers try to find which points are visible from a precise viewpoint but in our case, as different scanners can see the same points, we want to know for each point which scanner best sees it. We introduce a custom point cloud visibility method that serves our purpose and works well with LiDAR point clouds, regardless of the sampling density.

ScanFinder and PointCloudVisibility are two contributions which serves mainly the same purpose: expand *ContextCapture*^{TM} input point cloud formats. Another improvement made in *ContextCapture*^{TM} is point cloud compression. *ContextCapture*^{TM} provides a cloud service which gives the opportunity for people not having any clusters or high-performance machine to do the job; reconstructing a large surface requires effective machines. The problem is that, point clouds can be very huge, up to one hundred (100) gibabyte and more. And if something happens while uploading, the upload restarts from scratch. Being able to divide by two point cloud sizes and then reduce uploading time is an interesting point for *ContextCapture*^{TM} cloud services. Point cloud compression can be adressed in two ways: geometric compression [GKIS05, SK06] or pure arithmetic compression regardless of the kind of file being compressed. We compared different compressor such as Brolti, LZMA (7Zip), Zip before integrating one of them into the product.

This report is organised as follows. Chapter 2 present Bentley Systems, Acute3D, *ContextCapture*^{TM} and how the achieved work is positioned in the company's business line. After introducing in Chapter 3 some useful definitions for a better understanding of the report, we describe the achieved work of Scan Finder, Point Cloud Visibility and Point Cloud Compression respectively in Chapter 4, Chapter 5 and Chapter 6. Note that in each chapter, we recall the context, the issue addressed and the expected

[3]There is no need to know exactly which scanner generated it.

result before going into details. Finally Chapter 7 summarizes all the work, evaluates my contributions to *ContextCapture*$^{\boldsymbol{TM}}$ and assess what this experience has brought to me.

Chapter 2

The company

This chapter shed more light on *ContextCapture*^{TM} , the software I contributed to during these six (6) months internship as well as Bentley Systems, the company.

2.1 Bentley Systems

Bentley Systems is an American-based software development company founded by Keith A. Bentley and Barry J. Bentley in 1984.

For a bit of history[1], they introduced the commercial version of PseudoStation in 1985, which allowed users of Intergraph's VAX systems to use low-cost graphics terminals to view and modify the designs on their Intergraph IGDS (Interactive Graphics Design System) installations. Their first product was shown to potential users who were polled as to what they would be willing to pay for it. They averaged the answers, arriving at a price of $7,943. A DOS-based version of MicroStation was introduced in 1986. Later the two other brothers joined them in the business. Today, Bentley Systems is considered to have four (4) founders: Greg Bentley (CEO), Keith A. Bentley (EVP, CTO), Barry J. Bentley, Ph.D. (EVP) and Raymond B. Bentley (EVP).

At its core, Bentley Systems is a software development company that supports the professional needs of those responsible for creating and managing the world's infrastructure, including roadways, bridges, airports, skyscrapers, industrial and power plants as well as utility networks. Bentley delivers solutions for the entire lifecycle of the infrastructure asset, tailored to the needs of the various professions –

[1] Derived from [ben]

the engineers, architects, planners, contractors, fabricators, IT managers, operators and maintenance engineers – who will work on and work with that asset over its lifetime. Comprised of integrated applications and services built on an open platform, each solution is designed to ensure that information flows between workflow processes and project team members to enable interoperability and collaboration.

Bentley's commitment to their user community extends beyond delivering the most complete and integrated software – it pairs their products with exceptional service and support. Access to technical support teams 24/7, a global professional services organization and continuous learning opportunities through product training, online seminars and academic programs define their commitment to current and future generations of infrastructure professionals.

With their broad product range, strong global presence, and pronounced emphasis on their commitment to their neighbors, Bentley is much more than a software company – they are engaged functioning members of the global community. Their successes are determined by the skills, dedication, and involvement of extraordinary Bentley colleagues around the world.

Bentley has more than 3,500 colleagues in over 50 countries, and is on track to surpass an annual revenue run rate of $700 million. Since 2012, Bentley has invested more than $1 billion in research, development, and acquisitions.

2.2 Acute3D

Acquired by Bentley Systems in February 2015, Acute3D is now developing *ContextCapture*$^{\boldsymbol{TM}}$, as part of Bentley Systems' Reality Modeling solutions.

Acute3D is a technological software company created in January 2011 by Jean-Philippe Pons and Renaud Keriven, by leveraging on 25 man-years of research at two major European research institutes, École des Ponts ParisTech and Centre Scientifique et Technique du Bâtiment. It won the French "most innovative startup" Awards. In 2011, Acute3D signed an industrial partnership with Autodesk, while keeping to advance its R&D work on city-scale 3D reconstruction. In 2012, Acute3D signed industrial partnerships with other industry leaders, including Skyline Software Systems and InterAtlas. In parallel, it started to commercialize its own Smart3DCapture® (now replaced by *ContextCapture*$^{\boldsymbol{TM}}$) standalone software solution, optimized for highly detailed and large-scale automatic 3D reconstruction

from photographs. In 2015, Acute3D is acquired by Bentley Systems, and becomes part of their end-to-end Reality Modeling solutions.

2.3 *ContextCapture™*

With *ContextCapture™* , you can quickly produce even the most challenging 3D models of existing conditions for infrastructure projects of all types. Without the need for expensive, specialized equipment, you can quickly create and use these highly detailed, 3D reality meshes to provide precise real-world context for design, construction, and operations decisions for use throughout the lifecycle of a project.

Hybrid processing in ContextCapture enables the creation of engineering-ready reality meshes that incorporate the best of both worlds – the versatility and convenience of high-resolution photography supplemented, where needed, by additional accuracy of point clouds from laser scanning.

Develop precise reality meshes affordably with less investment of time and resources in specialized acquisition devices and associated training. You can easily produce 3D models using up to 300 gigapixels of photos taken with an ordinary camera and/or 500 million points from a laser scanner, resulting in fine details, sharp edges, and geometric accuracy.

Extend your capabilities to extract value from reality modeling data with ContextCapture Editor, a 3D CAD module for editing and analyzing reality data, included with ContextCapture. ContextCapture Editor enables fast and easy manipulation of meshes of any scale as well as the generation of cross sections, extraction of ground and breaklines, and production of orthophotos, 3D PDFs, and iModels. You can integrate your meshes with GIS and engineering data to enable the intuitive search, navigation, visualization, and animation of that information within the visual context of the mesh to quickly and efficiently support the design process.

Chapter 3

Basic Concepts

This chapter introduces the necessary concepts and definitions for a better understanding of the report.

3.1 Point Cloud

Definition 3.1 *: Point*

A point p_i is a tuple $\langle x_i, y_i, z_i, \vec{n}_i \rangle$ where:

- *i is an unique integer identifying p_i,*
- *x_i, y_i and z_i are the coordinates of p_i,*
- *$\vec{n}_i$ is the normal at p_i.*

Definition 3.2 *: Point Cloud*

A point cloud P of size s is a set $P = \{p_i \mid i \in [0, s]\}$

3.2 Some algorithms

3.2.1 Principal Component Analysis (PCA)

PCA is a common algorithm of data analysis. It is used to project data with n dimensions into a space with reduced dimensions while keeping the most relevant information – the directions where there is

the most variance, where the data is most spread out. As its name suggests, it finds the principal components from the most important to the less one. We say that the first principal component is a rotation of x-axis to maximize the variance of the data projected onto it and the last axis (component) is the one with less variance, with redundant information.

We are not going to explain in detail what a PCA is as it is very common and not the core of our work here. But let us remind the methodology.

Definition 3.3 *: Principal Component Analysis (PCA)*
Let P_a be a set of points of size s_a. To perform PCA(P_a):

- *we compute the centroid of P_a, noted $\bar{p_a}$*
- *we compute the covariance matrix C which is the symetric 3×3 positive semi-definite matrix:*

$$C = \sum_{p \in P_a} (p - \bar{p_a}) \otimes (p - \bar{p_a}), \text{ where } \otimes \text{ denotes the outer product vector operator.}$$

- *we compute the eigenvalues of C with their corresponding eigenvectors. The eigenvector $\vec{v}_1$ associated with the greatest eigenvalue λ_1 is the principal component axis. And the axis having less variation of data, when it is projected onto it is $\vec{v}_3$; the one associated with the lowest eigenvalue λ_3.*
- *we return a pair of $\langle \mu, \vec{v}_3 \rangle$ where $\mu = \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}$ can be used as a confidence level. The smaller it is, the more $\vec{v}_1$ and $\vec{v}_2$ explain on their own our data.*

In [HDD+92], Hoppe et al. use PCA to determine the normal of a tangent plane. Similarly in this report, we use PCA for two purposes: find the normal's direction of a plane and find the normal's direction at a point. If a set of points forms a plane, we choose its normal to be either $\vec{v}_3$ or $-\vec{v}_3$. It is the direction with less data variation and in a perfect word, that is a perfect plane alignment, the projection of all points onto it gives the same value. This means that $\mu = 0$. Now if P_a is formed of p_i and its k-neighbouring points, then $\vec{v}_3$ or $-\vec{v}_3$ is the normal at p_i. We empirically deduced fifty (50) to be the ideal value for k.

3.2.2 Least Square Methods

Least Square solving is a usual approach of regression analysis to approximate the solution of overdetermined systems (with more equations than unknowns). The problem can be described as follows. Let us assume a model of the form

$$y = f(z; x_1; ..., x_n) \tag{3.1}$$

where f describes a relation between $x_1, ..., x_n$, z is the control variable and y is the expected response to z. After m experiments, $(m \geq n)$, m observed quantities $(z_i, y_i), i \in [1, m]$ are collected. The purpose therfore is to find the paramaters $x_1, ..., x_n$ so that all z_i, y_i satisfy at best (3.1) . "Least squares" means that the overall solution minimizes the sum of the squares of the residuals made in the results of every single equation. The most important application is in data fitting. The best fit in the least-squares sense minimizes the sum of squared residuals, a residual being: the difference between an observed value, and the fitted value provided by a model.

Least-squares fall into two categories: linear and non-linear least squares, depending on whether or not the residuals are linear in all unknowns.

Linear

To solve linear least square problems, we use the Eigen library [GJ+10]. Consider an overdetermined system of equations, say $Ax = b$. Although it has no precise solution, it makes sense to search for the vector x which is closest to being a solution, in the sense that the difference $Ax - b$ is as small as possible. This x is called the least square solution (if the Euclidean norm is used). Provide Eigen with A and b and it will approximate the solution x.

Non-linear

In case of non-linear least square problems, we use the Ceres Solver [AMO]. Ceres Solver is an open source C++ library for modeling and solving large, complicated optimization problems. Ceres can solve bounds constrained robustified non-linear least squares problems of the form:

$$\min_x \quad \frac{1}{2} \sum_i \rho_i \left(\|f_i(x_{i_1}, ..., x_{i_k})\|^2 \right), \text{ such that } l_j \leq x_j \leq u_j$$

Ceres solver considers the expression $\rho_i \left(\| f_i(x_{i_1}, ..., x_{i_k}) \|^2 \right)$ as a *Residual Block*, where $f_i(.)$ is a *Cost function* that depends on the parameter blocks $[x_{i_1}, ..., x_{i_k}]$. We only need to specify the parameters, write the *Cost function* and Ceres does the job.

3.2.3 RANSAC

3.3 Geometric operations

3.3.1 Translation

3.3.2 Rotation

3.3.3 Projection ??

Chapter 4

Scan Finder

This chapter describes *Scan Finder*, an algorithm that can be used to retrieve all scanners position of a point cloud, if there are any, regardless of its nature. Firstly, Section 4.1 reviews the context which brings this need and describes some characteristics of the expected solution. Then, Section 4.2 reminds that there is no previous work related to this subject. Finally, Section 4.3 describes a first step toward scanner location finding before showing and discussing the results of two different approaches described respectively in Section 4.4 and Section 4.5. To be clear, Section 4.3 is a common first step to both methods.

4.1 Specifications

4.1.1 Context

ContextCapture^{TM} started to support point cloud reconstruction two (2) years ago. Today it even provides a hybrid processing mode which gives the opportuniy to combine the best of both words, photos and point clouds, in order to have a better precision. However, a recurring problem observed among *ContextCapture*^{TM} users is the impossibility to use the software after losing some metadata, specifically, scanners location. This is particulary problematic because usually, *ContextCapture*^{TM} users subcontract point cloud production to private companies which can charge them again for new exports (provided that they still have the point clouds). To enable customers to use their *defective* point clouds, the graphic interface of *ContextCapture*^{TM} allows to specify a scanner location by hand by positioning it in a 3D representation of the point cloud. But, it does not work with merged scans;

only one scanner location can be specified. Usually, users do not know the location and even if they know, the reconstruction is subjected to too much errors.

This is how the need for an algorithm to automatically find scanners positions in a point cloud is born. In addition, with such algorithm, *ContextCapture*^{TM} will have the possibility to enhance the set of supported file formats. Currently, it supports file formats such as *PTX*, *e57*, *PLY*, *POD*, each of them being able to store scanners positions. But not all file formats are able to do it. For instance, *LAS* file format is not currently accepted as input because it does not provide any means of storing scanners location in metadata. Supporting more input file formats is also a good point of interest for *ContextCapture*^{TM} users.

4.1.2 Objective

In summary, the purpose here is to improve *ContextCapture*^{TM} point cloud support in two ways: support more file formats and allow users who lose scanners location to still use their point clouds. To do so, the algorithm must:

- take as input any 3D point cloud captured from static scanners,
- be invariant to the number of scanners in the point cloud,
- be invariant to differences between scanners, such as: density, noise, rotation angle,
- find all scanners locations,
- have a reasonnable running time.

Let us emphasize here that as a first step, the algorithm is expected to work only with static point clouds. It would be difficult to have the same approach with static and mobile point clouds. Extending it to mobile point clouds is certainly the next step.

4.2 Related Work

As said in the introduction, to the best of our knowledge, there have been no previous work on the subject.

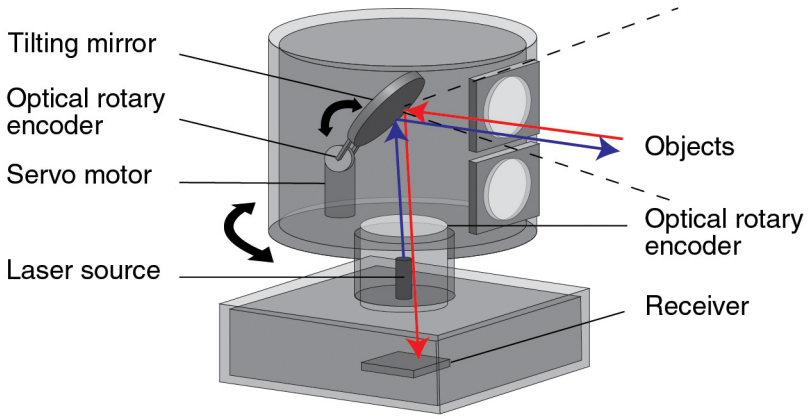


Figure 4.1: Example of a LiDAR scanner.

"Detecting the positions of multiple scanners exclusively from a point cloud is a subject not identified as interesting by academics, and thus not tackled since this information is almost always available from the outset. The same holds true on the industry side, it is only recently that scanner position information is relevant for a few applications like ContextCapture™ *."*

The closest publications on the topic are [TM15, SQLG15, KGC15, KC15, NS17]. They use learning techniques to estimate the point of view of one particular photo based on other photos. But this belongs more to the photogrametry domain and therefore is not applicable in a point cloud context.

4.3 Clustering high-density area

This section explains a common first step to methods described in Section 4.4 and Section 4.5. For a better understanding, a quick introduction to LiDAR scanners is necessary.

4.3.1 LiDAR scanner and context

LiDAR is an acronym of Light Detection and Ranging. It is a remote sensing technology which uses the pulse from a laser to collect measurements. The principle is simple: it works in a similar way to Radar and Sonar but uses light waves from a laser, instead of radio or sound waves. A LiDAR system calculates how long it takes for the light to hit an object or surface and reflect back to the scanner and then, uses the velocity of light[1] to calculate the distance. LiDAR systems can fire around 1,000,000

[1]The velocity, or speed of light is 299,792,458 metres per second.

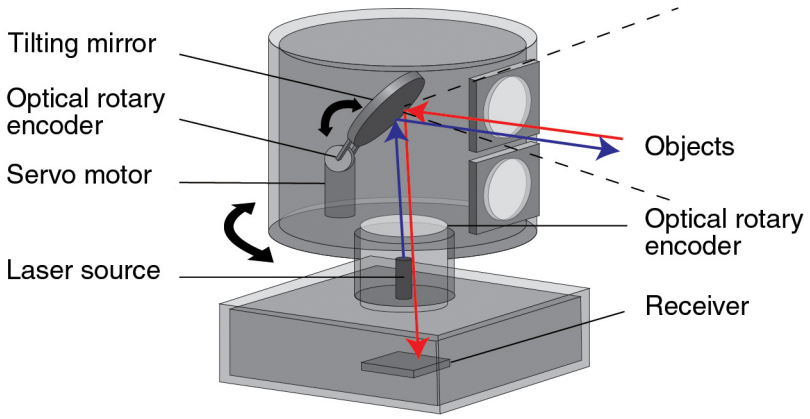


Figure 4.2: Variation of density as we go further away from the source.

pulses per second. This paragraph is inspired by [lid].

When scanning, there are two kind of rotations that a LiDAR scanner performs. They are indicated by black arrows in Figure 4.1. Not only a LiDAR scanner has a constant rotation angle when turning on itself but it also has a constant vertical rotation angle when scanning the environment. Because of these constant rotation angles, the further we go, the lower the point density is. On the contrary, the closer to the scanner we are, the higher the point density is. This density variation can be observed in Figure 4.2. To conclude, extracting the most dense areas is interesting as they always are around scanner locations and then, reduce the global search area.

4.3.2 Clustering and dense area extraction

The first step of the algorithm is to detect high density regions in the point cloud. For each point, we compute its density based on the mean distance to its k neighbors ($k \in [15, 50]$). We extract high density regions by selecting a percentage of the densest points (percentage between 1 and 5%). Figure 4.3 shows some results of this high density point extraction. As you can see, they always contain the scanner. Note that the scanner is not able to scan below himself leading to an elliptic form[2] appearing in all point clouds.

Once these points are extracted, a clustering step is applied. The clustering is needed so that points from the point clouds corresponding to potentially different parts of the scene around the scanner can

[2]This elliptic form is the key point of the method described in Section 4.5.

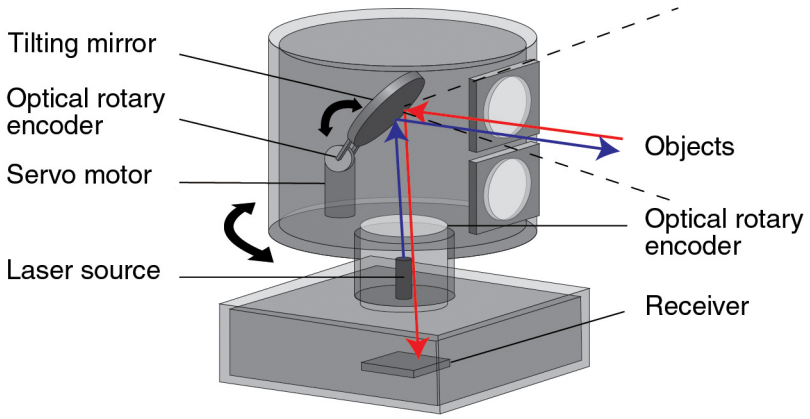


Figure 4.3: Result of high density extraction on several point clouds.

be separated. For each point, we compute its non-oriented normal and the associated *confidence level* by performing a principal component analysis (PCA) on its local neighborhood. Section 3.2.1 gives more details on how this PCA works. Then, as long as points are not attributed to a cluster, we select the point on the flattest surface, and iteratively add its neighbors that share a coherent normal orienta-tion (the dot product of the two normals must be superior to 0.8 in our experiments). Note that the flattest surface is determined using condifence levels, they tell which point belongs to a planar region.

Once every point is clustered, we discard clusters with too few points (in our experiments, we use a threshold of 0.2% of the number of high density points). We then detect clusters that are circular. A cluster is considered circular if its centroid, i.e. the average of all the points it contains, is far away from the points of the cluster.

At this point, we have computed a coarse location of the scanner. Methods described in Section 4.4 and Section 4.5 use it in their own way in order to find scanner locations.

4.4 The grid-pattern method

This section explains one approach we tried in order to solve the problem. Before explaining the method in detail, let us first give the intuition behind this idea.

4.4.1 Overview

As said in Section 4.3.1, LiDAR scanners perfom two kind of rotations. These constant vertical and horizontal rotation angles reveal some grid patterns on surfaces perpendicular to the ground[3]. Figure 4.4 shows an example of a grid pattern that can be found in point clouds. The purpose here is to find, for a single grid pattern, a relation between all constant rotation angles and the scanner's position. Therefore, this relation can be applied with all possible grid patterns in the point clouds, leading to a problem with a huge set of constraints to solve. The idea is that only the real scanner's location is able to explain in the best way these constant rotation angles.

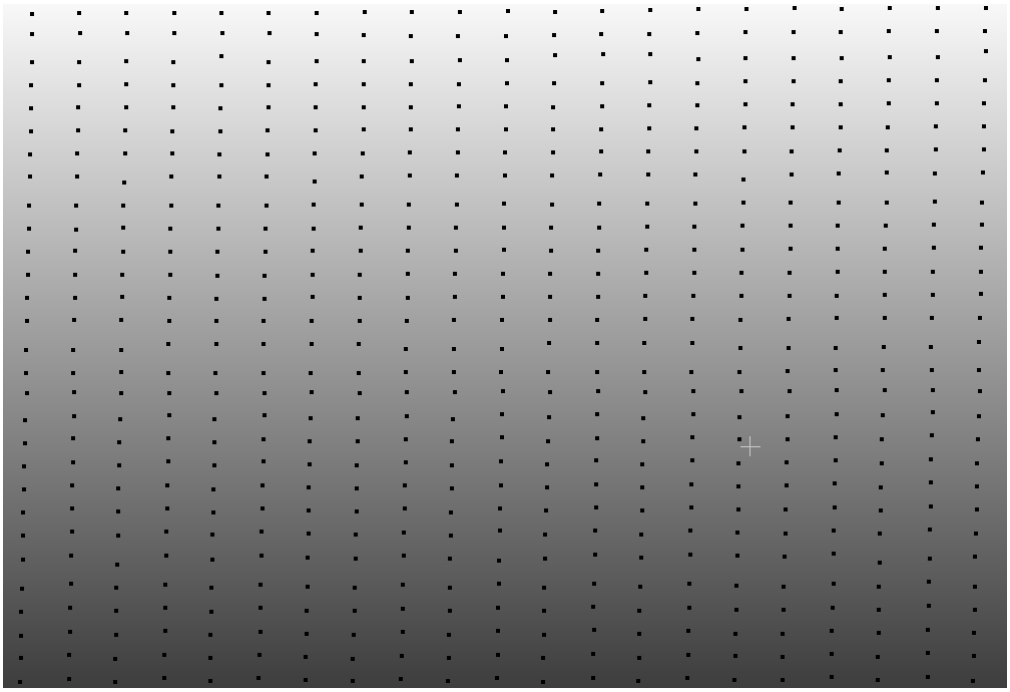


Figure 4.4: Example of a grid pattern. As you can see there is a constant offset between each vertical line and each horizontal lines. Note that the offset between columns is not necessary the same than between lines.

This algorithm can be divided into two parts. The first step is to find *accurate* grid patterns in point clouds. Once this is done, the second step is to build the equation that will be solved. These two parts are explained in the following subsections. Note that this approach does not work in multiscan mode, compare to the other approach described in Section 4.5. It assumes there is only one scanner which explains all grid-patterns in the point cloud.

[3]To be precise, perpendicular to the surface on which the scanner is installed.

Function `GetPlanarPatches(`P`)`
- **Input:** a pointcloud $P = \{p_i \mid i \in [0, s]\}$.
- **Output:** a set of potential grid-pattern patches.
- /* We compute the normal of each point and keep both confidence level and normal $\langle \mu_i, \vec{v}_i \rangle$. */
- /* See Section 3.2.1 for more details on PCA algorithm. */
- /* From here, $\forall i \in [0, s]$ we assume $\vec{v}_i$ to be the normal at i and μ_i its confidence level. */
- **for** ($i = 0;\ i < s;\ i = i + 1$) {
  - $\langle \mu_i, \vec{v}_i \rangle \leftarrow PCA(50 \text{ closest neighbours of } i)$
- $R \leftarrow \emptyset$
- visited $\leftarrow \emptyset$
- **for** ($i = 0;\ i < s;\ i = i + 1$) {
  - /* Continue if already visited or is not planar enough. */
  - **if** $i \in$ visited or $\mu_i > 0.001$ **then**
    - **continue**
  - tmp $\leftarrow \{p_i\}$
  - **foreach** $j \in 50$ closest neighbours of i **do**
    - /* If $\vec{v}_j$ and $\vec{v}_i$ are almost colinear and j has not been seen yet. */
    - **if** abs$(\vec{v}_j \times \vec{v}_i) > 0.9$ and $j \notin$ visited **then**
      - tmp $\leftarrow$ tmp $\cup\ p_j$
      - visited $\leftarrow$ visited $\cup\ j$
  - /* If there is enough point for a grid-pattern patch. */
  - **if** size of tmp $> 0.9 \times 200$ **then**
    - $R \leftarrow R \cup$ tmp
- **return** R

Algorithm 1: Find various not-overlaping planar patches in a point cloud.

4.4.2 Grid-pattern matching

This subsection explains how to find *accurate* grid patterns in point clouds in order to reduce as much as possible the noise and its impact during the problem resolution described in Section 4.4.3. A set of point is considered as an *accurate* grid-pattern when:

- it is planar as much as possible[4],
- the underlying planar surface is orthogonal to the surface of the ground,
- the gap between columns is consistent enough,
- the gap between lines is consistent enough.

Algorithm 1 shows how to extract some planar patches in a point cloud. A preprocessing step already done and explained in Section 4.3.2 is to compute for each point of the point cloud, the normal and its *confidence level*. This *confidence level* plays a key role in the algorithm as it tells how planar the

[4] Because of the noise.

```
Eigen::VectorXf FitLine(std::vector<A3D::Point_3dPlus> const& points)
{
  Eigen::MatrixXf mat(points.size(), 2);
  Eigen::VectorXf vec(points.size());
  int count = 0;
  for (auto it = points.begin(); it != points.end(); ++it)
  {
    mat(count, 0) = static_cast<float>(it->x());
    mat(count, 1) = static_cast<float>(it->y());
    vec(count++) = 1.0;
  }
  return mat.fullPivHouseholderQr().solve(vec);
}
```

Figure 4.5: Function using least-square to fit a line to the given set of points.

region around each point is. By browsing all points, each time we find a point with a good *confidence level* (> 0.001), we start to build a patch around it. To do so, we extract its 50 closest neighbours and consider only those having their normal almost colinear to the normal of the starting point. If the size of the obtained set is bigger enough ($> 0.9 \times 200$), the set is kept and otherwise, not. Also, in order to avoid overlaping or patches too close, we keep track of visited points.

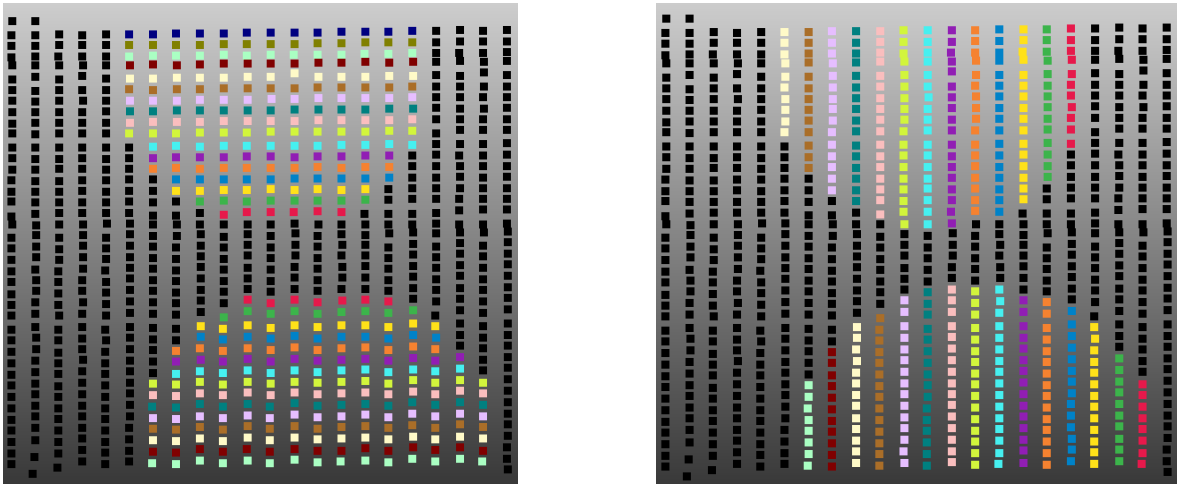
Although planar, this set of patches potentially contains: **(a)** patches whose underlying surfaces are not orthogonal to the surface of the ground and **(b)** patches without grid patterns. To filter out each **(a)** patch, we use a dot product between its normal and the normal of the circular cluster found in the high-density area[5] (see Section 4.3). For **(b)** patches, the objectif is to indentify grid-patterns. To do so, we use least square method in order to fit lines. Least-square is explained in Section 3.2.2. Fitting a line fall into linear least square problems. Figure 4.5 shows how to fit a line to a set of points using the Eigen library [GJ+10]. The purpose is to use this line fitting in order to cluster lines and columns. Figure 4.6 shows results of a clustering performed on two patches after several fittings and adjustements. We only keep firstly patches containing a grid-pattern and then, patches having a regular gap between columns and lines. The rest is discarded.

Once only *accurate* patches are kept, columns and lines are identified, the problem can be formalised.

4.4.3 Equation to solve

For a little recall of Section 4.4, the purpose here is to find a relation between the scanner's position (what we are looking for) and the regular gaps between lines and columns. Again, the gap between

[5]The circular cluster desribes the surface on which the scanner is positioned.



(a) Lines clustering performed on two patch.

(b) Column clustering performed on two patches.

Figure 4.6: Example of columns and lines clustering in order to identify grid-patterns.

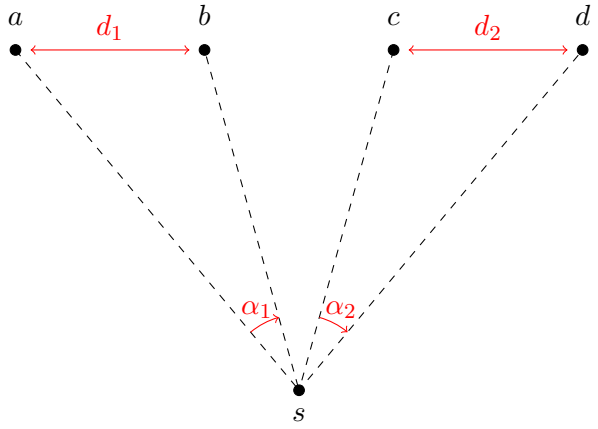


Figure 4.7: A top view over four (4) vertical grid lines (a, b, c and d) and a scanner (s).

lines and the gap between columns is not necessary the same. Also, at this point, we already have a reduced area of research which is arround the circular cluster obtained in Section 4.3.2. What we want, is a set of equations that will help to set the point within this area.

Look at Figure 4.7. There are four (4) vertical lines viewed from a bird's eye: a, b, c and d. They are represented as points because of the point of view. The regular gap between those lines is represented by the relation $d_1 = d_2$. But instead of working on the regular gap distance, one can work with the regular rotation angle of the scanner. These angles directly involve the scanner location in all equations. Here is how the problem is built: for each grid-pattern, and for each pair of two consecutive lines (or columns), we minimize the absolute difference between their respective angles (here α_1 and α_2). This is a non-linear least-square problem.

FIXME: More explanation?

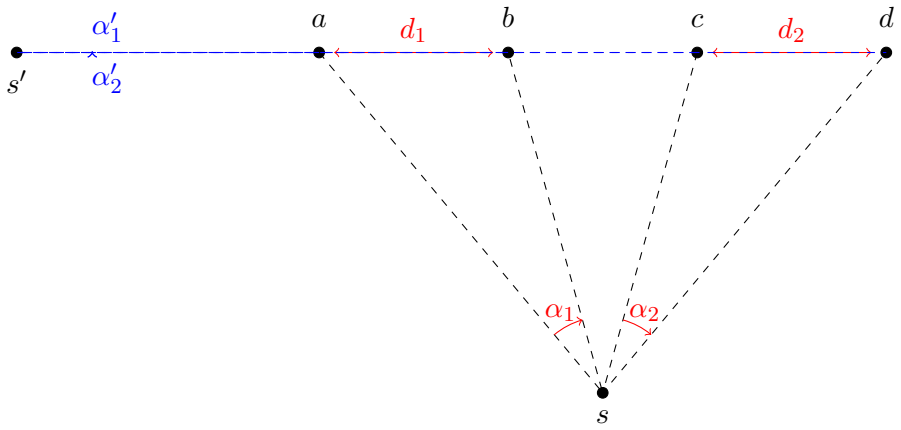


Figure 4.8: Illustration of the problem preventing the grid-pattern method from finding the scanner location. Not that this is a top view over four (4) vertical grid lines (a, b, c and d), the apprixmation (s') and the real scanner (s).

4.4.4 Results and discussions

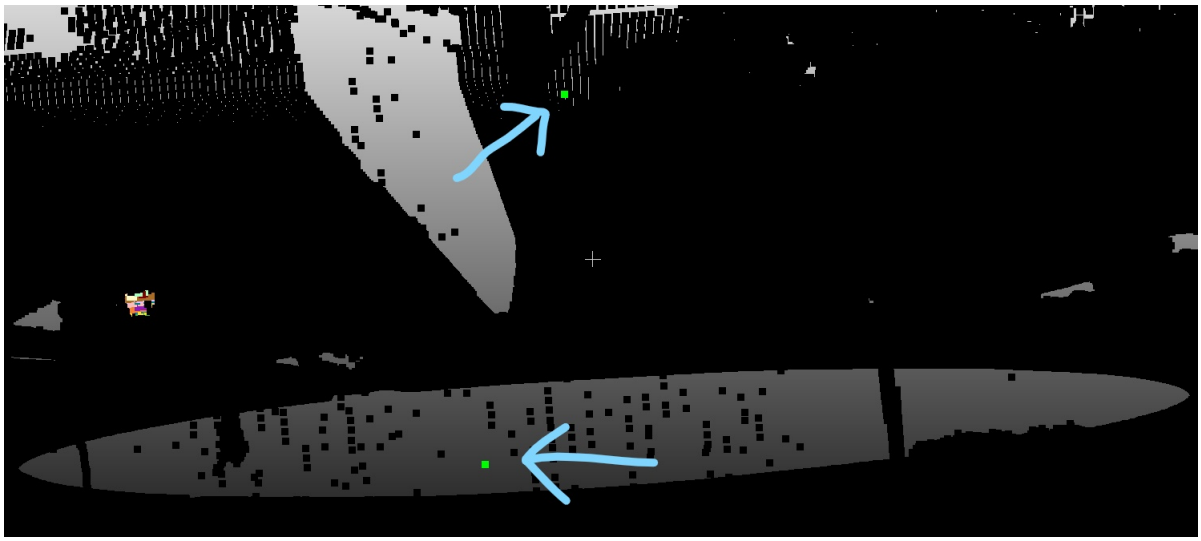


Figure 4.9: An approximation of the scanner location using the grid-pattern method. The real scanner location is highlighted by the upper arrow while the approximation is highlighted by the lower one.

Figure 4.9 shows a result of grid-pattern approximation of scanner location. This method does not perform quite well. One would think that the approximation is not far from the real scanner location but reducing the area of research to a cube arround the circular cluster is somewhat misleading. We tried to remove the area constraints and observed that the approximation moves entirely away from the real location and even the point cloud itself. It can stoop pretty low, go to really high altitude, on the left, on the right. It completely depends on the distribution of the grid-patterns in the point cloud.

Let us take one grid-pattern in order to illustrate the problem: Figure 4.8. As our method tries to minimize the differences between all pairs of angles, here α_1 and α_2, the ideal case is if the scanner location belongs to the underlying surface of the grid-pattern. Here, α'_1 and α'_2 are perfectly equal: $\alpha'_1 - \alpha'_2 = 0$. Therefore, with a huge set of equations involving several grid-patterns, the optimal solution for the solver is to put the scanner in the underlying surface of the *mean plan of all grid-pattern underlying planes*. This is why, without the reduced area constraint, the approximated scanner is far away from the point cloud as the solver tries to reduce all angle differences and then, satisfy all grid-patterns.

To conclude, angles differences are not discriminating enough. We believe there is a way to express the problem in another way, in order to bypass this behaviour but we deciced to try another approach presented in Section 4.5. In addition, this approach is limited to one scanner whereas the other method works with multiple scaners.

4.5 The elliptic method

4.5.1 Overview

4.5.2 Fitting ellipse

4.5.3 Equation to solve

4.5.4 Results and discussions

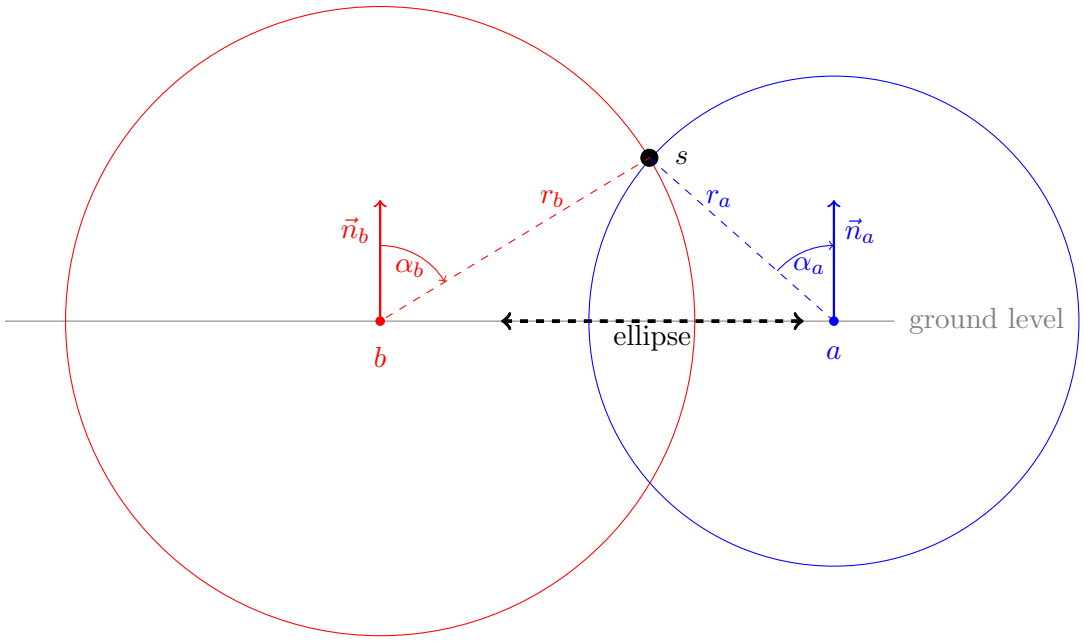


Figure 4.10: A side view of the ellipse and two spheres intersecting at the scanner position s and their respective centers a and b. We have the relation $\frac{d_a \times r_a^2}{\cos(\alpha_a)} = \frac{d_b \times r_b^2}{\cos(\alpha_b)}$.

Chapter 5

Point Cloud Visibility

5.1 Specifications

5.1.1 Problem being addressed

5.1.2 Objective

5.1.3 Scope

5.2 Related work

5.3 Direct Visibility of Point Sets

5.3.1 Overview

5.3.2 Implementation

5.3.3 Results and discussions

5.4 Visibility of Noisy Point Cloud Data

5.4.1 Overview

5.4.2 Implementation

5.4.3 Results and discussions

Chapter 6

Point Cloud Compression

6.1 Specifications

6.1.1 Problem being addressed

6.1.2 Objective

6.1.3 Scope

6.2 Related work

6.3 A custom arithmetic approach

6.3.1 Overview

6.3.2 Implementation

6.3.3 Comparison with Brotli, 7Z and Zip

6.4 Integration

Chapter 7

Conclusion

7.1 Summary of Internship Achievements

Summary.

7.2 Applications

Applications.

7.3 Future Work

Future Work.

Bibliography

[AMO] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[ben] Bentley systems by wikipedia. `https://en.wikipedia.org/wiki/Bentley_Systems#History`. Accessed: 2018-07-06.

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[GKIS05] Stefan Gumhold, Zachi Kami, Martin Isenburg, and Hans-Peter Seidel. Predictive point-cloud compression. In *ACM SIGGRAPH 2005 Sketches*, page 137. ACM, 2005.

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. *Surface reconstruction from unorganized points*, volume 26. ACM, 1992.

[KC15] Alex Kendall and Roberto Cipolla. Modelling uncertainty in deep learning for camera relocalization. *arXiv preprint arXiv:1509.05909*, 2015.

[KGC15] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[lid] What is lidar? `https://www.3dlasermapping.com/what-is-lidar-and-how-does-it-work/`. Accessed: 2018-07-06.

[NS17] Yoshikatsu Nakajima and Hideo Saito. Robust camera pose estimation by viewpoint classification using deep learning. *Computational Visual Media*, 3(2):189–198, 2017.

[SK06] Ruwen Schnabel and Reinhard Klein. Octree-based point-cloud compression. *Spbg*, 6:111–120, 2006.

[SQLG15] Hao Su, Charles R Qi, Yangyan Li, and Leonidas J Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2686–2694, 2015.

[TM15] Shubham Tulsiani and Jitendra Malik. Viewpoints and keypoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1510–1519, 2015.