|  |
| --- |
| ENSTA Bretagne |
| UV5.4 Status Report |
| Pierre Jacquot – SPID/ROB |

|  |
| --- |
|  |



Supervised by :

* Pierre Bosser
* Nathalie Debèse

Summary

[Introduction 3](#_Toc444161341)

[Context 3](#_Toc444161342)

[1. Pre-processing 3](#_Toc444161343)

[1.1 Point Cloud Artifacts 4](#_Toc444161344)

[1.2 Normal Estimation 5](#_Toc444161345)

[2. Surface Function 7](#_Toc444161346)

[2.1 Signed Distance Function (4) 7](#_Toc444161347)

[2.2 Indicator function 8](#_Toc444161348)

[3. Surface reconstruction 9](#_Toc444161349)

[3.1 Marching Cube Algorithm 9](#_Toc444161350)

[3.2 Delaunay and Voronoï triangulation 10](#_Toc444161351)

[Conclusion 11](#_Toc444161352)

[. References (10) 12](#_Toc444161353)

[Figure 1 - Graphical representation of the different forms of point cloud artifacts (3) 4](#_Toc444161237)

[Figure 2 - Part of the Kereon scan. The white points on the right of the windows are considered as noise, and are created by the laser going throught the window 4](#_Toc444161238)

[Figure 3 - missing data on Kéréon scan 5](#_Toc444161239)

[Figure 4 - Oriented normals of a cloud point (6) 5](#_Toc444161240)

[Figure 5 - Representation of the two first eigenvectors from a covariance matrix of a given dataset 6](#_Toc444161241)

[Figure 6 - Comparison between a bad orientation (on the left) and a good orientation (on the right) of the normals 7](#_Toc444161242)

[Figure 7 - Principle of the marching cube algorithm 9](#_Toc444161243)

[Figure 8 - The fifteen triangulated cubes 10](#_Toc444161244)

[Figure 9 - Index creation for the marching cubes algorithm 10](#_Toc444161245)

[Figure 10 - Voronoï Diagram. Each red dot represent the center of the cell 11](#_Toc444161246)

# Introduction

Computers are constantly increasing in term of power, efficiency and capacity. This quick evolution allow us to manage more and more data at the same time. This ability to deal with a large amount of data permit now to deal with what we called cloud points. This point cloud are the results of laser scans, and basically contain a collection of data, each containing the coordinates and sometimes the colors of what the laser has scanned. This ability to represent our world using point cloud have many applications. We can for example use this to recreate architectural site, or to recreate an environment in prevision of a future operation [1] or even into the biomedical field.

However when we talk about recreating a monument, or an object using point cloud, this also mean that we have to create a 3D version of this model. To do so we have to do a surface reconstruction of the object or monument. This surface reconstruction imply to link each point, in a logical way with each other in order to obtain an accurate 3D reproduction of the desired object.

As you can imagine such a task imply many pre-requisites. Among them we first need to understand how the 3D point cloud has been acquired, what kind of object we want to reconstruct (an exterior, an interior, a simple object?). Linking a point cloud also need pre-treatment, so we can obtain the most accurate representation. We’ll therefore need to clear the point cloud of any noise or misplace points and simplify it if it contains too many points, so we can have a light mesh (a mesh is a collection of vertices, edges and faces that defines a shape[2]). Here it will be the final 3D representation of the building we’re trying to recreate). Furthermore linking the points between them is not trivial as there is not necessary a right order to do it, and the computer certainly don’t know in advance which order will be the best. We will have to recreate the surface in an implicit way. Therefore, the goal of this status report is to explain step by step how to reconstruct the surface of a cloud point, by pointing out in a chronological way the best methods from pre-processing the point cloud to reconstructing its surfaces. You’ll find in the following sections an explanation of the most used and robust methods to do so.

# Context

This project is collaboration between les Phares et Balise (a department of le parc marin d’Iroise) and the ENSTA Bretagne. Les Phares et Balises are currently trying to put forward some of the lighthouses of the Finistère’s coast. These lighthouses are for most of them too far away from the coast and despite their strong cultural interest cannot be visited. To tackle this issue, les phares et balises have organized several laser scans of these lighthouses so that people could visit them. The main idea is to present a 3D representation of these lighthouses (focusing on the lighthouse of Kereon) during the Brest 2016 festival. As meshing a cloud point is not trivial they ask for the ENSTA Bretagne expertise to create a 3D mesh of the lighthouse of Kereon.

# 1. Pre-processing

Recreating a mesh from a raw point cloud is not trivial and need a good understanding on the data we’re using and on the challenges we’ll have to face. Therefore, we’ll be focusing on this next part on the first obstacle will have to overcome before meshing our point cloud. Thus we will talk about artefacts that can appear in your raw data, (mostly because of the scan quality), [3] , and the normal associated with each point [4]. Each of this part will be a determinant factors for the surface reconstruction methods.

## 1.1 Point Cloud Artifacts

Laser scanning an area often comes with many non-wanted features appearing in the point cloud. These unwanted features are called artifacts. The most impactful on the surface reconstruction are: the sampling density, the noise, the outliers, the misalignment and the missing data [4]. All of these artifacts will be explained in the next parts and we’ll be dealt with later, during the surface reconstruction part.

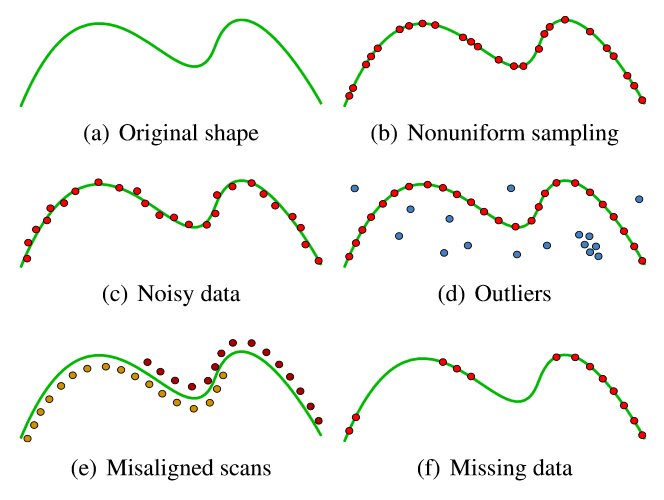


Figure 1 - Graphical representation of the different forms of point cloud artifacts [4]

**Non uniform sampling:**

Non uniform sampling is visible on figure 1(b). This king of artefacts is in majority due to the positioning of the scanner regarding the object or scene we are scanning. Other factors impacting point sampling are the orientation of the scanner and also the shape of the objects we are scanning. A good way to tackle this issue is to scan an object multiple time, and with various angle in order to have the right amount of points.

**Noise:**

One of the most common artefact. Noise is due to many factors, including the sensor of the scanner, the distance and orientation of the surface scanned and the inner characteristic of the surface scanned. For example reflective surfaces are a major source of noise as well as windows (figure 2). You can either try to eliminate noise (which can be fairly easy on the example figure 2), or you can produce a surface that passes near the noise or ignore it.

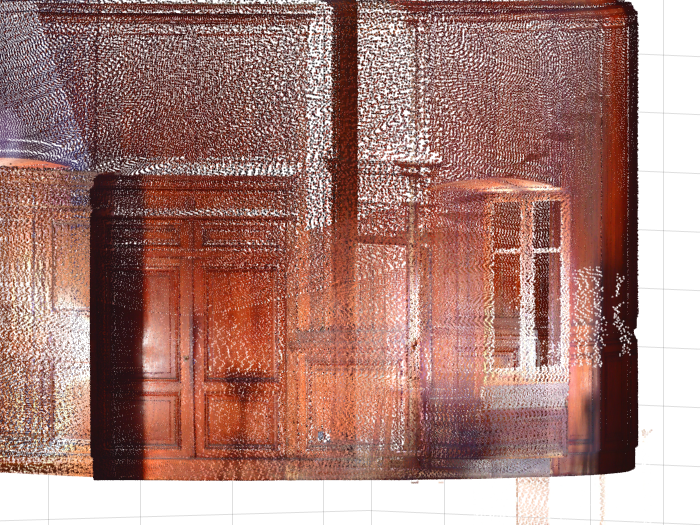


Figure 2 - Part of the Kereon scan. The white points on the right of the windows are considered as noise, and are created by the laser going throught the window

**Outliers:**

The outliers are the points far from the true surface. These artifacts are due to structural artifacts in the acquisition process. This type of artifact often appear in multi-view stereo acquisition when points taken with a different angles result in false correspondences. This is important to note that outliers must not be taken into account in the surface reconstruction and must be detected and erased.

**Missing data:**

Missing data are due to limited sensor range, high light absorption and occlusions in the scanning process. To avoid this kind of problem multiple scanned must be done in order to overlap them, reducing the quantity of missing data, but causing sometimes misaligned scans (figure 1(e)). In the case of the Kéréon scans we can see some missing data located on the floor area (figure 3), where the scanner was laid.

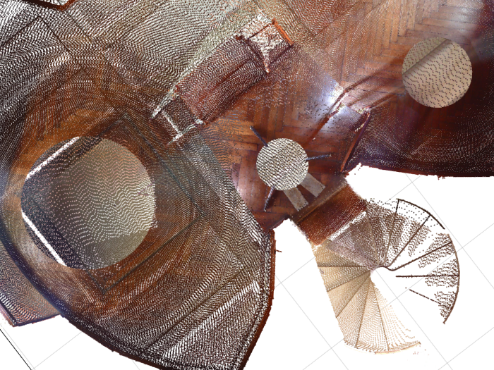


Figure 3 - missing data on Kéréon scan

## 1.2 Normal Estimation

Surface normals are really important input to some reconstruction methods such as the Poisson methods [5] that we are going to explain. We are calling normals, the normal to the tangent plane associated with a data point (cf figure 4). As a matter of fact, finding all tangent plane is a method to reconstruct the surface of the point cloud as each tangent plane is a localized part of the final surface. However we’ll explain this in another part. We’ll be focusing here on finding the normal to each point and how to correctly orient them.

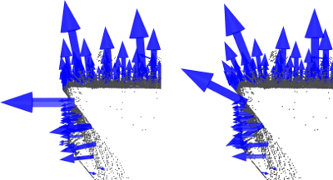


Figure 4 - Oriented normals of a cloud point [6]

Let’s define the tangent plane associated with the data point and represented by is center the point and its associated unit normal vector [4]. We defined the center as the centroid of the closest neighbor (where is user-specified) of . This set of neighbor is denoted as . To compute the we will be using the covariance matrix of defined as follow

Let’s now denote the eigenvalues of CV and the associated eigenvectors. Then, using Principal Component Analysis (7) we can approximate or to be the unit normal vector of the tangent plane associated to the data point . As a matter of fact the eigenvectors of the covariance matrix give information about the pattern of the data[7]. The eigenvector associated with the highest eigenvalue will represent the line where the data are the more correlated. At the opposite the eigenvector associated with the lowest eigenvalue will represent the line where the data are the less correlated. And the last eigenvector will represent a less important correlation of the data. For example the figure 5 shows the two first eigenvectors of a strongly oriented set of points. Knowing that each eigenvector are perpendicular to each other, we can conclude that the two first eigenvectors will be included in the tangent plane and the third one will be the normal to this plane.

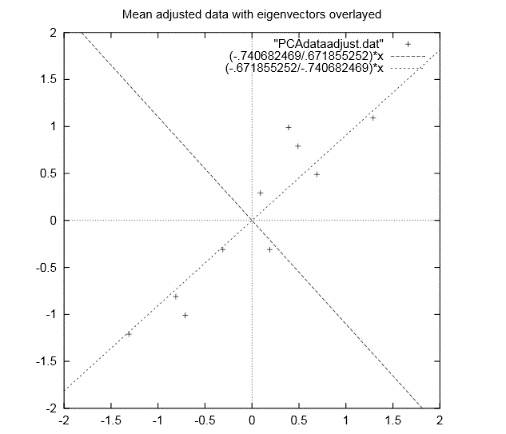


Figure 5 - Representation of the two first eigenvectors from a covariance matrix of a given dataset [7]

Now that we have our normal vectors to each data point, we should ideally oriented them, i.e. make them all point toward the inside of the surface or toward the outside. By doing so we can understand better if we are inside or outside the surface we are trying to reconstruct. This will be also useful later on in one of the algorithm we will describe.

A natural way to easily orient all the normals will be to give the same orientation to points close in a geometric point of view. However this kind of orientation in not robust when the surface we’re considering has sharp angles. As the figure 6 shows, two points can be close but can have really different normals orientations.

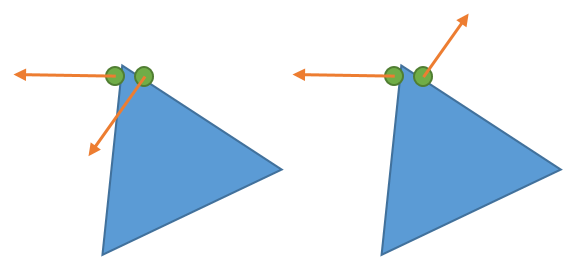


Figure 6 - Comparison between a bad orientation (on the left) and a good orientation (on the right) of the normals

In order to solve this problem [4] we will be using the value where and represent the normal vectors associated with two close centroid and . So if then it means that the two normals are parallel and have the same orientation. Thus the ideal way to orient the normal is to propagate the orientation following the path were is always at is minimum.

Furthermore orienting normals can also be done using the coordinates of the scanner. Indeed the scanner will always be inside the surface we’re scanning or outside, and can therefore easily gives the normals orientations. However this imply to note the scanner position during the scan.

# 2. Surface Function

This part will be focusing on the definition of the surface function of the object we want to recreate. A surface function is not to be confused with surface reconstruction. The surface reconstruction aim to recreate the surface of the point cloud, meaning to link all the points in a coherent way to create a polygonal mesh. However the surface function, will be a scalar function that define the surface, i.e. can tell us whether or not a point is located on the surface. The surface function can be used to define an isosurface. An isosurface is an implicit function defined as follow:

[8]

can be seen as a threshold delimiting the surface and is called the isovalue. Basically if then we are at the exterior of the surface and if we are inside the surface. Therefore here defines our isosurface but is also the surface function we are looking for. We will present to you two methods to determine this function in the section below.

## 2.1 Signed Distance Function [4]

This part will use the same notations as the part 1.2. However we will here define to be a sampled data point vector approximating the surface . Furthermore each vector of can be defined as follow:

Here represent the inaccuracy of the laser, thus will be the error. The sample is called -noisy.

is also -dense, meaning that any sphere with a radius of and center in contains at least one sample point in .

The signed distance function can be defined as follow:

The sign of this function will give us on which side of the surface the point is located. However this signed distance is not totally the surface function we are searching as it will not give us the belonging or not of the point to the surface . To define the surface function, [4] used the fact that is -dense and -noisy. Therefore the projection of a point on will always be at a distance inferior to . Thus, the surface function can be defined using the algorithm:

This structure function allow to easily recreate a surface, however it is not really robust to the noise especially for small value of were the noise hide the true nature of surface by not giving the right centroid for the tangent plane. On the opposite taking bigger value for will result in a less precise mesh. The key factor is to correctly determine to obtain the best representation possible. The next part will present a more robust surface function using the Poisson problem to be computed.

## 2.2 Indicator function

The indicator function is used to compute the surface function used in the Poisson surface reconstruction [5], one of the easiest and most efficient method to do surface reconstruction. The indicator function is defined as follow:

The indicator function, is therefore really useful to describe a surface. However finding it is not trivial. To do so we’re going to use the oriented vectors we learnt to compute in the 1.2 section.

As a matter of fact, the gradient of will be equal to 0 everywhere, except on the surface of the model we are studying by definition of the indicator function (indeed is nearly constant everywhere). Thus we can say that where represent the inward surface normals. Then we can transform this problem in a Poisson problem by applying the divergence operator. Thus, finding the indicator function will be possible by solving this equation:

Solving this Poisson problem offer some advantages, one being that it takes into account all the points at the same time. This advantage allow a really robust 3D reconstruction of the surface to the noise. This robustness is allowed by the surface function we are using to extract our isosurface.

The surface function will be defined as follow (using the same notation as before):

Here the isovalue will be . The isovalue allow a really robust to the noise reconstruction because it takes into account all the points of the surface, therefore diminishing the impact of the noise on the 3D surface.

This surface function is not only reluctant to noise but also offer really good result in term of resolution. Using the Poisson surface reconstruction is really an efficient way to obtain good results, even with non-uniform and non-oriented data.

# 3. Surface reconstruction

The final step to the creation of our mesh is the surface reconstruction. The next part we’ll describe two classical methods for surface reconstruction (i.e. creating a polygonal mesh from our model).

## 3.1 Marching Cube Algorithm [9]

The marching cube algorithm is defined by Lorensen and Cline as a march and conquer algorithm. Taking as input data the surface function it is able to give a triangle mesh as output. The idea is to decompose the 3D space of the model in a 3D grid compose of cubes. Each cube of this grid is therefore contain between two slices as shown on the figure 6.

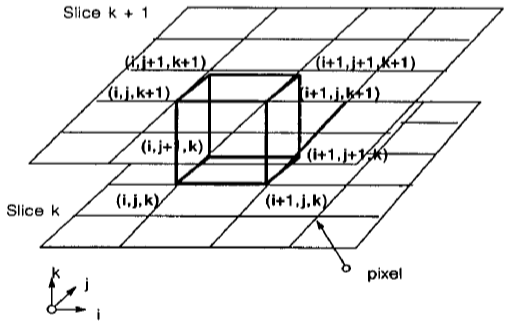


Figure 7 - Principle of the marching cube algorithm

Then the algorithm is quite simple. For each vertex of the cube we apply the structure function. If the result is below the isovalue then we assigned a 0 to this vertex. If the value is above the isovalue we assign a 1 to this vertex. Thus the surface we trying to recreate will intersect an edge of the cube if the vertices associated with this edge are respectively at 0 and 1.

However since there are 8 vertices, will be counting ways a surface can intersect the cube. However using the symmetry and rotation properties of the cube we can reduce these 256 cases to only 15 shown below.

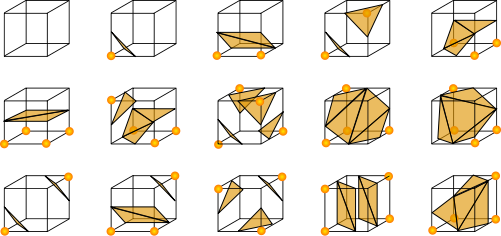


Figure 8 - The fifteen triangulated cubes

An easy way to compute this algorithm is to associate with each vertex of the cubes one byte, so we can easily represent each case by creating an 8 bits index. (cf. Figure9)

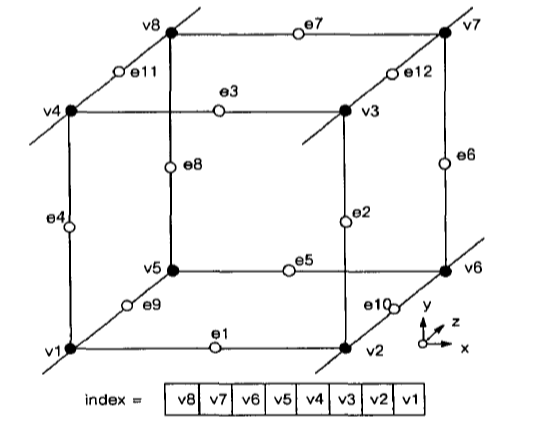


Figure 9 - Index creation for the marching cubes algorithm

The marching algorithm is really a widely used, easy to compute algorithm (it is used both in [4] and [5]). This algorithm offer really fast and good result and can be adapt to any case scenario. Combine with a smoothing algorithm it gives really good results.

## 3.2 Ball Pivoting Algorithm [10]

The marching cubes algorithm is not the only using a standard geometry in order to guess and produce a surface from cloud point. As a matter of fact the ball pivoting algorithm uses a ball, or more precisely a sphere to do the surface reconstruction of the mesh. One of the major advantages of this method is that you don’t need a surface function to make it works as it only uses the positions of the point cloud’s samples.

Let’s still consider as a sampled, and -dense data set of the surface . Let’s now consider a sphere of radius . Now if we put this ball in contact with the surface it will never goes through as it will always be in contact with at least 2 points. Let’s now consider the triangle , where are the three edges of the triangles (and three points of X) and are contained into the sphere or radius . This triangle can be considered as the first piece of our surface and is called the seed. Finally we will note the edge of the triangle linking and .

Considering these notations the algorithms works as followed:

Starting with the sphere containing the three edges of the triangle we’re going to make this sphere rotate around the edge . If the sphere encounter another point then another triangle is created. However if no point is encountered the edge is therefore considered as a boundary. Some special cases exists when the sphere encounter a point already contained into the surface. These cases are well explained in [10] and will not be explained here. The algorithm continue until all edges are either contained in the surface or boundaries. The figure below illustrate the algorithm in 2D.

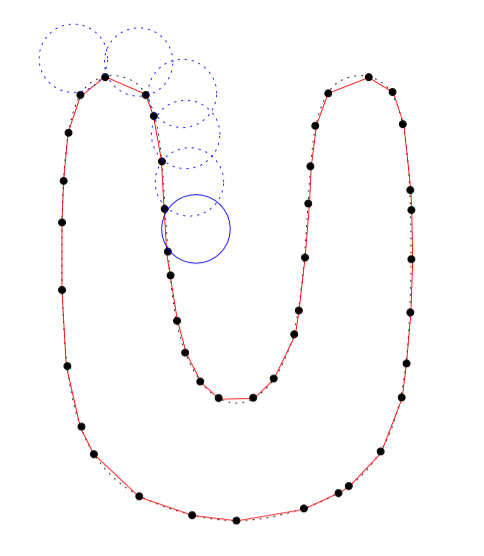


Figure 10 - Ball pivoting algorithm [10]

Ones of the major advantages of this algorithm is that it is really fast and reluctant to noise. This noise robustness is due to the fact that the ball will often not touch the noisy point as they will be outside the general surface as shown on the figure X:

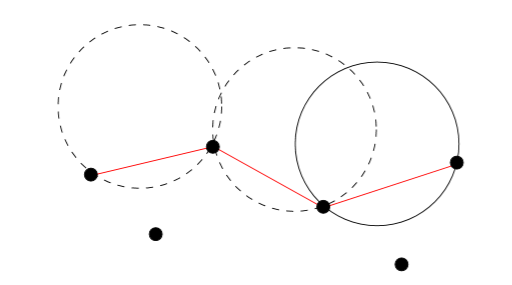


Figure 11 - Ball Pivoting algorihtm in presence of noiy data [10]

In case that a noisy point is taken into account a simple way to get rid of it is to compare the normals of the three edges and the normal of the faces created by the algorithm. If they are not collinear the algorithm will erase this triangle from the final surface.

However the algorithm can encounter some problem if the sampled data have some missing data. Thus the ball will “fall” trough the surface when pivoting (see figure X). However tackling this issue can be done by applying the ball pivoting algorithm several time with different radius for the ball.

Another major problem will occur when the diameter of the ball is too big to fit in a part of the surface. As a matter of fact if the curvature of the surface is larger than then all the sample point will not be taken into account (see figure X). This is why it is extremely important to know the density of your cloud point before applying the ball pivoting algorithm.

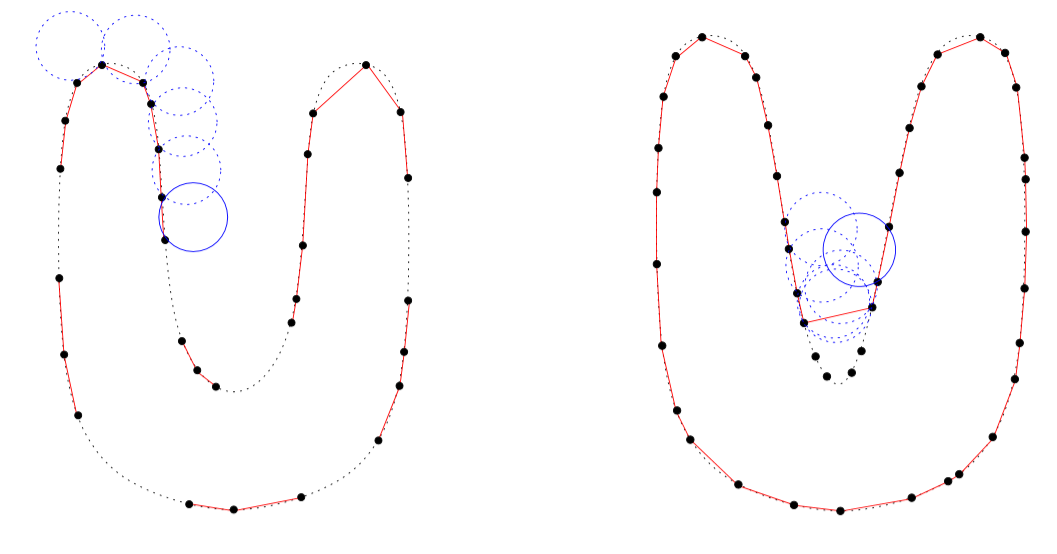


Figure 12 - Biggest issues encounter with the ball bivoting algorithm

As a conclusion the ball pivoting algorithm is a robust and easy to use algorithm that will be tested during the course of my project.

## 3.3 Delaunay and Voronoï triangulation

Other methods of polygonal meshing exist that doesn’t use the structure function. For example the Crust algorithm uses the Voronoï diagram and the Delaunay triangulation to compute the polygonal mesh [11]. This algorithm first decompose the point cloud in a Voronoï diagram. This Voronoï is composed of multiples cells where a cell associated with a vertex is defined as follow:

Finding all the cells contain in the cloud point will thus create the Voronoï diagram shown in figure 10.

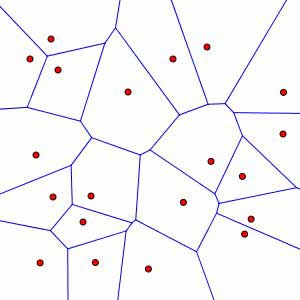


Figure 13 - Voronoï Diagram. Each red dot represent the center of the cell

We can then use the Delaunay triangulation, combine with the diagram method to create a polygonal mesh of our point cloud. The criteria for the Delaunay Triangulation is that for each triangle composing the mesh, the circumscribe circle associated with the triangle should not contain other point of the point cloud rather than its vertices.

This is where the Voronoï diagram is useful. As a matter of fact each center of the Voronoï diagram is a center of a circumscribe triangle used in the Delaunay triangulation. Using the combination of these two techniques, the Crust algorithm is able to compute the polygonal mesh.

This method is interesting as it doesn’t need to know a lot of things about our point cloud except the coordinates of each points. However it’s not robust to noise as it does a really local reconstruction, taking into account each point. Furthermore this method really depends on the number of points contained in our input point cloud. It can really fast be time consuming. It’s however a good method to quickly create a polygonal mesh of small point cloud, without a lot of noise. However, even if the concept is interesting this algorithm will not be tested for our surface reconstruction because the poisson and ball pivoting algorithms besides using some of its methods are much more efficient and mostly used for surface reconstruction.

# 4. First Conclusion

As a conclusion it appears that doing surface reconstruction from a cloud point is a pretty difficult task especially when the cloud point considered has not a lot to offer in terms of information. This status report will therefore be really useful in my case as I’m working on a pretty complicate and dense point cloud. The Poisson surface algorithm and ball pivoting seem to be a good way to start as they seem pretty robust and widely used by different software.

The next step will be the implementation of this algorithm. A good start will be with MeshLab or with the C library GCAL because they already offer the tools to do such classical methods and can handle large amount of data easily.

Associated with the reconstruction it will also be interesting to work on a way to simplify the cloud point I’m working with at it is quite a big one.

Finally, another task to do will be to map the colors and textures of the lighthouse onto the polygonal mesh I will have recreated. This mapping of colors will be challenging as I have quite a lot of missing data in my point cloud and should whereas be dealt with.

The next parts of this will report will focus on the realization of a 3D reconstruction of the Kéréon’s lighthouse.

# 5. Realization

The parts that are going to follow will now talk about the concrete realization of the lighthouse in 3D and a creation of a virtual visit. They will focus on giving a methodology to reconstruct efficiently a room of the lighthouse. We will first talk about the goal of this project then of the data I used for the reconstruction, and I’ll finally present the different software I used and methods I apply to have the best representation. Given the nature of data (specified in the next section) most of my work has been empirical.

## 5.1 Goals and limitations

This project goal was to create a 3D representation of the Kéréon’s lighthouse that can be presented during a festival and/or on “les phare et balises” website. The main idea was to be able to visit in an interactive way the lighthouse. We decided to do an interactive visit in a 3D environment for the website and a video that can be shown during the festival. However considering the amplitude of the task I decided to focus only on two rooms in order to describe a methodology that can be applied on each other room to recreate the lighthouse in its entirety. Therefore the next sections will chronologically explained the nature of the data, how to process the cloud point, how to create the 3D mesh and finally create an interactive visit and a video of the lighthouse.

## 5.1 Raw data

The data I’ve been working on during this project were given by “Les phares et balises” and were the result of a scan done using a Faro laser. The specification of the laser were not known to me. The data I had were the combination of several scans of the room from different position (in order to capture all the room). We can clearly see on the next figure the disposition of the laser during the three scans used to recreate the cloud point:

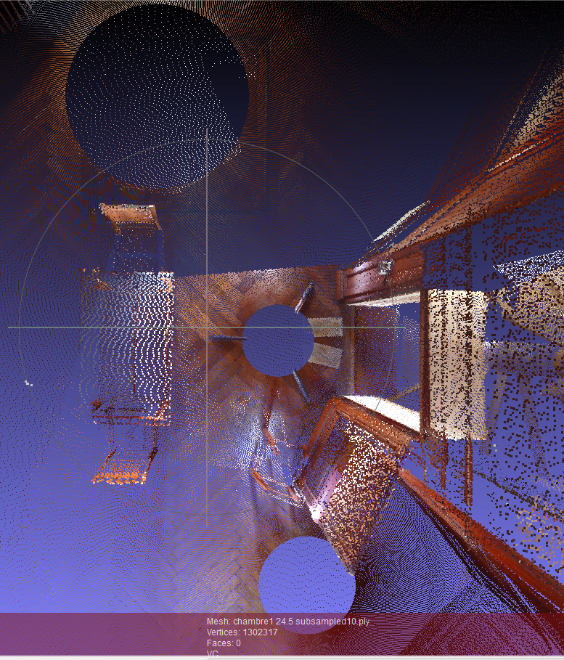


Figure 14 - Each circle in the cloud point represents the position of the laser during the scan

Each point cloud have been combined using JRCreconstructor a proprietary and non-free software. After being reconstructed, the clouds point have been subsampled 10 times and exported in the .ply format.

The ply format is a standard format for 3D object representation and stands for Polygon File Format. This format is widely used by 3D software as it is an easy way to store the coordinates of points resulting of a laser scan. However this format not only store in an easy to use list the coordinates of the points, it can also store their colors (in an RGB form), their normals or even their transparencies. The figure below shows how the data are stored in a classic ply files.

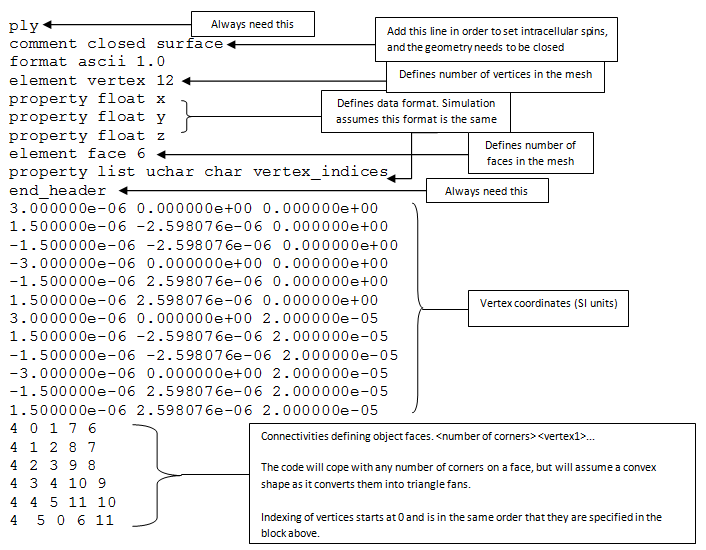


Figure 15 - Structure of a ply files were the coordinates of the vertex and their connectivities (i.e how they are linked to create faces) are stored. [12]

Using matlab to read the files given to me I determined that only the colors and the position of each point were given. Therefore the point clouds will need some pre-treatment before applying a surface reconstruction method on them.

The two rooms that we will be working on are similar in shape but have really different cloud point in term of density. The first one called chamber1 is composed of 1302317 points and the second called chamber2 is composed of 15224042 points. As we can see there are approximatively 10 times more point in the chamber 2 than in the chamber1.

## 5.2 Software

Before talking about the methodology itself it seems important to me to talk first about the different software I have been using and why I chose them

### 5.2.1 Meshlab

Meshlab is the main software I have used during my project. I chose for many different reasons. First of all it is a totally free software. Moreover it handles really well dense cloud point and more importantly all the methods I wanted to use were already develop and implemented in it. Thus I used this software for the surface reconstruction, and to export the mesh into Blender (that will be presented in the section 5.2.3).

### 5.2.2 Cloud Compare

Cloud compare is a really powerful software to manipulate big and dense cloud point. I didn’t use it to create the 3D mesh but to manually change the cloud point I was working on. As a matter of fact it offers great tools to easily manipulate and erase unwanted part of cloud point. For example, I removed some parts and furniture I didn’t want in my room as they were always giving bad results.

### 5.2.3 Blender

Blender is an open source 3D graphics and animation software that I decided to use to create the interactive visit and the video of the lighthouse. I chose Blender because I can easily create video games using the Blender game engine and can also create a video of a 3D model.

## 5.3 Cloud simplification

**Poisson reconstruction parameters**

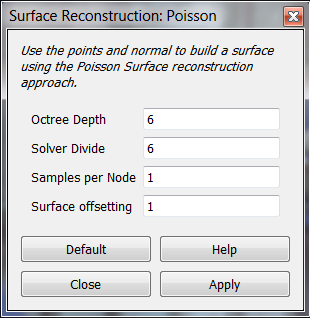


Figure 16 - Poisson reconstruction parameters in Meshlab

**Octree Depth:**

In order to understand this really important parameter it is important to know what an octree is.To solve in an efficient and quick way the Poisson equation and apply the marching cubes algorithm for their surface reconstruction, (6) had to discretize the cloud point. To do so they divide the point cloud using an octree. The octree is a way of structuring the space by regrouping all the point of your cloud into cubes. For example a cube containing all your point will be equivalent to an octree of depth 1. Divide this cube into 8 equals cubes (called octants) and you will obtained an octree of depth 2. By continuing this subdivision and increasing the depth of your octree you will therefore give each point more weight has it can finally be alone in an octant. The figure 12 illustrates the subdivision of a mesh using an octree of depth 4, 6 and 8.

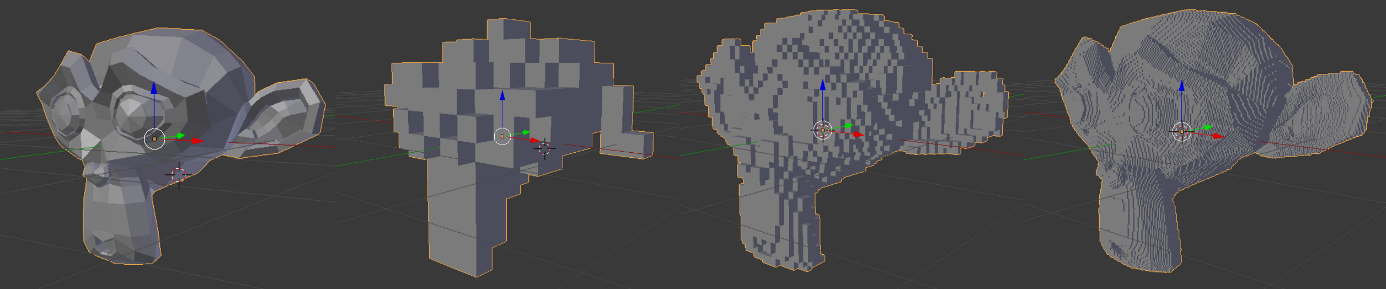


Figure 17 - From left to right: original mesh, octree depth 4, octree depth 6, octree depth 8

We can describe the structure and construction of an octree as followed:

* The cube of depth 1 is the first node of the tree and is called the root
* Each node as either 8 children or no children
* A node without children is called a leaf and contained the data we want to store
* A node with children is called an internal node
* An internal node is defined by its center and its width
* A node will become a leaf if it is empty or if we are at the maximum depth of the octree

Here the last property is quite important as it will stop the subdivision if we are in an area with no points and continue the subdivision in area with point and interesting feature. This mean that the area with more points will be more precisely reconstruct during the Poisson reconstruction. Another way to describe it is to say that having an octree depth of d, will give a volume resolution up to 2dx2dx2d.

**Solver Divide:**

The solver divide represent the depth at which the solver of the Poisson equation will be applied in the octree. This can be useful to reduce the computing time of the algorithm with high depth octrees.

**Samples per Node:**

Defines the number of sample contained in each nodes of the octree. This can be useful for a really noisy point cloud to smooth it. As a matter of fact the Poisson equation’s is solved for each node so putting more point per node will reduce the spatial resolution of a clean mesh but will smooth a noisy ones. Meshlab advices to put between 1 to 5 points for a normal cloud and between 15 and 20 for a noisy one.

**Surface offsetting:**

Allow to have an offset to the surface in case of a lack of accuracy during the laser scan. The default value 1.0 correspond to no offset.

# REFERENCES

[1] A. Bey, « Reconstruction de modèles CAO de scènes complexes à partir de nuages de points basée sur l’utilisation de connaissances a priori », 2012. [En ligne]. Disponible sur: http://liris.cnrs.fr/~rchaine/EDF\_A\_BEY/These-Aurelien-Bey.pdf. [Consulté le: 20-févr-2016].

[2] « Polygon mesh », *Wikipedia, the free encyclopedia*. 21-févr-2016.

[3] M. Berger, A. Tagliasacchi, L. Seversky, P. Alliez, J. Levine, A. Sharf, et C. Silva, « State of the art in surface reconstruction from point clouds », in *EUROGRAPHICS star reports*, 2014, vol. 1, p. 161–185.

[4] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, et W. Stuetzle, *Surface reconstruction from unorganized points*, vol. 26. ACM, 1992.

[5] M. Kazhdan, M. Bolitho, et H. Hoppe, « Poisson surface reconstruction », in *Proceedings of the fourth Eurographics symposium on Geometry processing*, 2006, vol. 7.

[6] « Documentation - Point Cloud Library (PCL) ». [En ligne]. Disponible sur: http://pointclouds.org/documentation/tutorials/normal\_estimation.php. [Consulté le: 24-févr-2016].

[7] H. Anton, *Elementary linear algebra*. New York: Wiley, 1987.

[8] R. Wenger, *Isosurfaces: Geometry, Topology, and Algorithms*. CRC Press, 2013.

[9] W. Lorensen et C. Harvey, « Marching Cubes: A High Resolution 3D Surface Construction Algorithm », in *Computer Graphics*, 1987, vol. 21.

[10] A. S. Glassner, International Conference on Computer Graphics and Interactive Techniques, et Association for Computing Machinery, Éd., *SIGGRAPH 94 conference proceedings: July 24 - 29, 1994, [Orlando, Florida]*. New York, NY: ACM, 1994.

[11] Y. Nasser et M. Ouyous, « Reconstruction de surfaces d’objets 3D a partir de nuage de points », 2015.

[12] « Camino | Monte-Carlo Mesh Simulation ». [En ligne]. Disponible sur: http://camino.cs.ucl.ac.uk/index.php?n=Tutorials.MCMeshSimulation. [Consulté le: 14-mars-2016].