

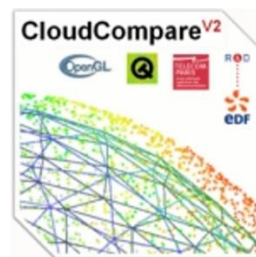
ENSTA Bretagne

# UV5.4 Creation of a virtual tour from a point cloud

Pierre Jacquot – SPID/ROB

Supervised by :

- Pierre Bosser
- Nathalie Debèse
- Michel Legris



## Summary

Introduction.....	3
Context .....	3
1. Pre-processing .....	3
1.1 Point Cloud Artifacts.....	4
1.2 Normal Estimation.....	5
2. Surface Function.....	7
2.1 Signed Distance Function [4] .....	7
2.2 Indicator function .....	8
3. Surface reconstruction .....	9
3.1 Marching Cube Algorithm [9] .....	9
3.2 Ball Pivoting Algorithm [10].....	10
3.3 Delaunay and Voronoï triangulation .....	12
4. First Conclusion .....	13
5. Realization .....	13
5.1 Goals and limitations.....	13
5.1 Raw data.....	13
5.2 Software .....	15
5.2.1 Meshlab .....	15
5.2.2 Cloud Compare .....	15
5.2.3 Blender .....	16
5.3 Normal computation .....	16
5.4 Cleaning .....	17
5.5 Surface Reconstruction .....	18
5.5.1 Poisson Algorithm.....	18
5.5.2 Ball Pivoting Algorithm .....	21
5.6 Color Generation .....	23
5.7. Creation of the virtual visit.....	24
6 Improvements and limitations .....	26
7 Conclusion .....	27
APPENDIX 1: Poisson Parameters (filter by number of faces) .....	28
APPENDIX 2 : Ball Pivoting Algorithm.....	29
APPENDIX 3 : Tutorials .....	31
REFERENCES .....	33

Figure 1 - Graphical representation of the different forms of point cloud artifacts [4] .....	4
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 through the window .....	4
Figure 3 - missing data on Kéréon scan .....	5
Figure 4 - Oriented normals of a cloud point [6] .....	5
Figure 5 - Representation of the two first eigenvectors from a covariance matrix of a given dataset [7] .....	6
Figure 6 - Comparison between a bad orientation (on the left) and a good orientation (on the right) of the normals .....	7
Figure 7 - Principle of the marching cube algorithm .....	9
Figure 8 - The fifteen triangulated cubes .....	10
Figure 9 - Index creation for the marching cubes algorithm .....	10
Figure 10 - Ball pivoting algorithm [10] .....	11
Figure 11 - Ball Pivoting algorithm in presence of noisy data [10] .....	11
Figure 12 - Biggest issues encountered with the ball pivoting algorithm .....	12
Figure 13 - Voronoi Diagram. Each red dot represents the center of the cell .....	12
Figure 14 - Each circle in the cloud point represents the position of the laser during the scan .....	14
Figure 15 - Structure of a ply file where the coordinates of the vertex and their connectivities (i.e how they are linked to create faces) are stored. [12] .....	14
Figure 16 - The Meshlab interface .....	15
Figure 17 - The CloudCompare interface .....	16
Figure 18 - The Blender interface .....	16
Figure 19 - Normal computation filter in Meshlab .....	17
Figure 20 - Normals of the chamber2 represented by blue arrows .....	17
Figure 21 - Quick reconstruction of the chamber2 to see the impact of the furniture on the mesh ...	18
Figure 22 - Poisson reconstruction parameters in Meshlab .....	18
Figure 23 - From left to right: original mesh, octree depth 4, octree depth 6, octree depth 8 .....	19
Figure 24 - On the left a reconstruction using an octree depth of 8 and on the right using an octree depth of 10 .....	20
Figure 25 - On the left: 15 samples per node, On the right : 1 sample per node .....	20
Figure 26 - On the left: the chamber1, on the right: the chamber2 .....	21
Figure 27 - Ball Pivoting parameters .....	21
Figure 28 - The ball pivoting algorithm applied on the chamber2 .....	22
Figure 29 - Result of the ball pivoting algorithm on the 6 parts of the point cloud .....	23
Figure 30 - On the left: Color of chamber2, on the right: Color of chamber1 .....	24
Figure 31 - Bounding Box in a video game .....	25
Figure 32 Player representation and bounding box for the lighthouse tour .....	25
Figure 33 - Logical brick in Blender .....	26

## Introduction

Computers are constantly increasing in terms of power, efficiency and capacity. This quick evolution allows us to manage more and more data at the same time. This ability to deal with a large amount of data permits now to deal with what we called cloud points. These point clouds 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 clouds has many applications. We can for example use this to recreate architectural sites, 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 means that we have to create a 3D version of this model. To do so that we have to do a surface reconstruction of the object or monument. This surface reconstruction implies to link each point, in a logical way with each other in order to obtain an accurate 3D reproduction of the desired object.

Such a task implies 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 needs pre-treatment, so we that 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 together is not trivial as there is not necessarily a right order to do it, and the computer certainly doesn'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 a collaboration between les Phares et Balises (a department of le Parc Marin d'Iroise) and 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 Brest 2016 International Maritime festival. As meshing a cloud point is not trivial they ask for 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 of 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 we 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 these parts will be a decisive factor 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 we will be explained in the next parts and we'll be dealt with later, during the surface reconstruction part.

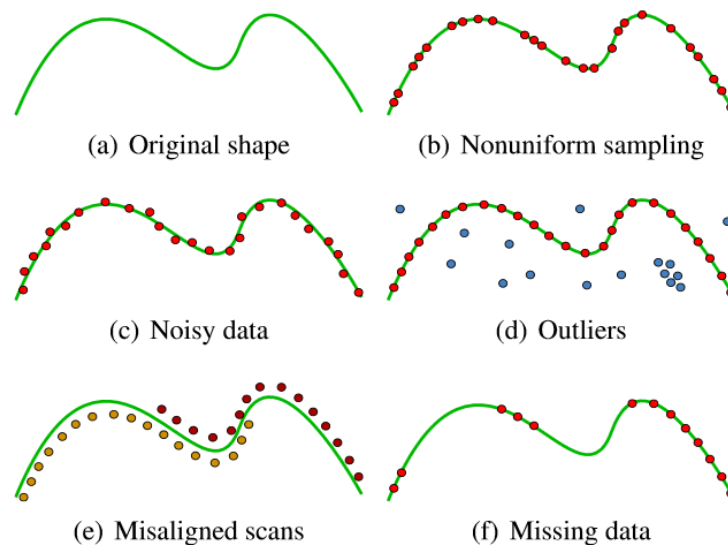


Figure 1 - Graphical representation of the different forms of point cloud artifacts [4]

### Non uniform sampling:

Non uniform sampling is visible on figure 1(b). This kind of artifacts 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 times, and with various angles 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 characteristics 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 ignores 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 through 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 artifacts often appears in multi-view stereo acquisition when points taken with a different angle result in false correspondences. It 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 scans 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.

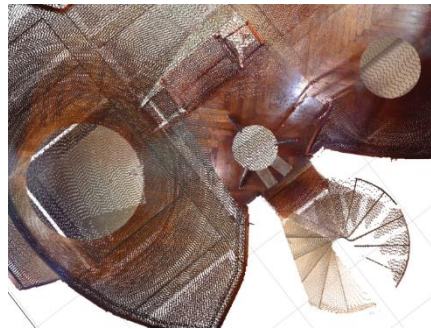


Figure 3 - missing data on Kéréon scan

## 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.

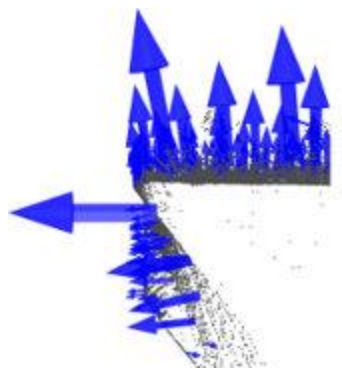


Figure 4 - Oriented normals of a cloud point [6]

Let's define  $T(x)$  the tangent plane associated with the data point  $x$  and represented by its center the point  $o$  and its associated unit normal vector  $\hat{n}$  [4]. We defined the center  $o$  as the centroid of the  $k$  closest neighbor (where  $k$  is user-specified) of  $x$ . This set of neighbor is denoted as  $Nbgh(x)$ . To compute the  $\hat{n}$  we will be using the covariance matrix of  $Nbgh(x)$  defined as follow

$$CV = \sum_{y \in Nbgh(x)} (y - o) \otimes (y - o)$$

where  $\otimes$  is the outer product vector operator

Let's now denote  $\lambda_1 \geq \lambda_2 \geq \lambda_3$  the eigenvalues of CV and  $v_1, v_2, v_3$  the associated eigenvectors. Then, using Principal Component Analysis (7) we can approximate  $v_3$  or  $-v_3$  to be the unit normal vector  $n$  of the tangent plane associated to the data point  $x$ . 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 most 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 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.

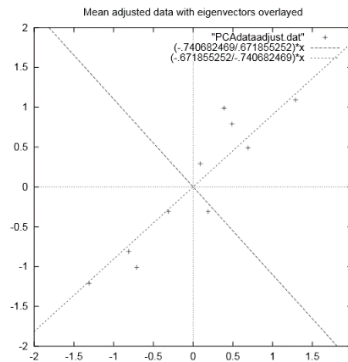


Figure 5 - Representation of the two first eigenvectors from a covariance matrix of a given dataset [7]

Now that we have our normal vectors to each data point, we should ideally orient 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 is not robust when the surface we're considering has sharp angles. As figure 6 shows, two points can be close but can have really different normals orientations.

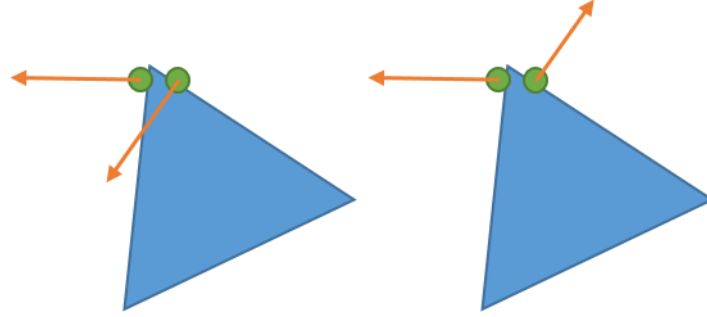


Figure 6 - Comparison between a bad orientation (on the left) and a good orientation (on the right) of the normals

In order to solve this problem [4] we will be using the value  $\alpha = 1 - |\hat{n}_i \cdot \hat{n}_j|$  where  $\hat{n}_i$  and  $\hat{n}_j$  represent the normal vectors associated with two close centroids  $o_i$  and  $o_j$ . So if  $\alpha = 0$  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 where  $\alpha$  is always at its 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 give the normals orientations. However this implies 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 aims 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 defines 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 follows:

$$F(x, y, z) = f \text{ where } f \text{ is a constant [8]}$$

$f$  can be seen as a threshold delimiting the surface and is called the isovalue. Basically if  $F(x, y, z) > f$  then we are at the exterior of the surface and if  $F(x, y, z) < f$  we are inside the surface. Therefore  $F$  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 part 1.2. However we will here define  $X$  to be a sampled data point vector approximating the surface  $M$ . Furthermore each vector of  $X$  can be defined as follow:

$$\forall x \in X, x = y + e, y \in M, \|e\| \leq \delta$$

Here  $\delta$  represents the inaccuracy of the laser, thus  $e$  will be the error. The sample  $X$  is called  $\delta$ -noisy.

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

The signed distance function can be defined as follow:

$$\forall p \in \mathbb{R}^3, f(p) = \text{dist}(p) = (p - o) \cdot \hat{n}$$



The sign of this function will give us on which side of the surface the point  $p$  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  $p$  to the surface  $M$ . To define the surface function, [4] used the fact that  $X$  is  $\rho$ -dense and  $\delta$ -noisy. Therefore the projection  $z$  of a point  $p \in \mathbb{R}^3$  on  $M$  will always be at a distance inferior to  $\delta + \rho$ . Thus, the surface function can be defined using the algorithm:

$$\begin{aligned}
 & z = o - ((p - o) \cdot \hat{n}) \cdot \hat{n} \text{ where } z \text{ is the projection of } p \text{ onto } T(x) \\
 & \quad \text{if } d(z, X) < \rho + \delta \\
 & \quad f(p) = \text{dist}(p) = (p - o) \cdot \hat{n} \\
 & \quad \text{else} \\
 & \quad f(p) = \text{undefined}
 \end{aligned}$$

This structure function allow to easily recreate a surface, however it is not really robust to the noise especially for small value of  $k$  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  $k$  will result in a less precise mesh. The key factor is to correctly determine  $k$  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  $\chi$  is defined as follows:

$$\begin{cases} \forall p \in \mathbb{R}^3, \chi(p) = 1 \text{ if } p \text{ is inside } M \\ \chi(p) = 0 \text{ if } p \text{ is outside } M \end{cases}$$

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 1.2 section.

As a matter of fact, the gradient of  $\chi$  will be equal to 0 everywhere, except on the surface of the model we are studying by definition of the indicator function (indeed  $\chi$  is nearly constant everywhere). Thus we can say that  $\nabla \chi = \vec{V}$  where  $\vec{V}$  represents 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:

$$\Delta \chi = \nabla \cdot \vec{V}$$

Solving this Poisson problem offers some advantages, one being that it takes into account all the points at the same time. This advantage allows 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 follows (using the same notation as before):

$$\forall p \in \mathbb{R}^3, p \in M \Leftrightarrow \chi(p) = \gamma, \quad \text{with } \gamma = \frac{1}{\text{card}(X)} * \sum_{x \in X} \chi(x)$$

Here the isovalue will be  $\gamma$ . The isovalue allows 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 offers really good result in terms 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 will 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 the surface function as an input data 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 composed of cubes. Each cube of this grid is therefore contained between two slices as shown in figure 6.

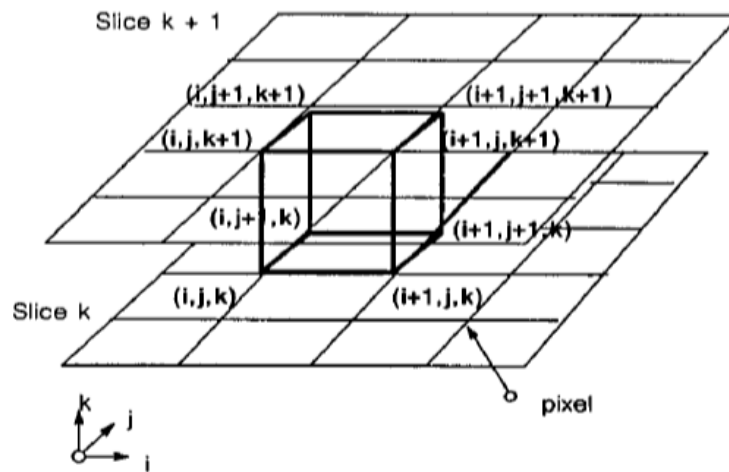


Figure 7 - Principle of the marching cube algorithm

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, we will have counting  $2^8 = 256$  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.

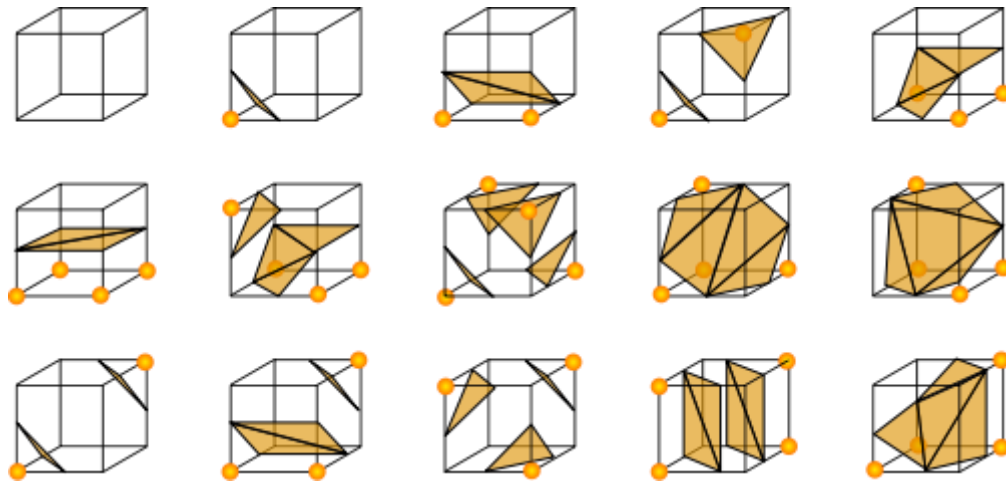


Figure 8 - The fifteen triangulated cubes

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

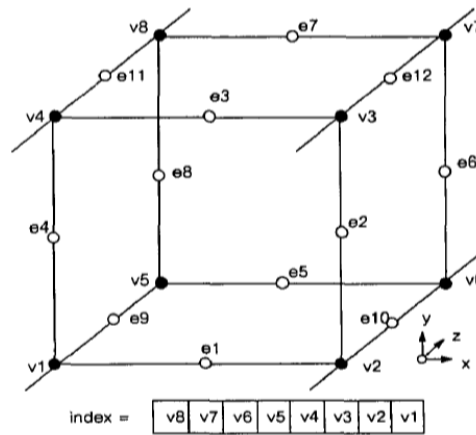


Figure 9 - Index creation for the marching cubes algorithm

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

### 3.2 Ball Pivoting Algorithm [10]

The marching cubes algorithm is not the only one using a simple geometry shape in order to guess and produces a surface from point clouds. As a matter of fact the ball pivoting algorithm uses a ball, or more precisely a sphere to do the surface reconstruction of the point cloud. 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 each vertex of the point cloud.

Let's still consider  $X$  as a sampled, and  $\rho$ -dense data set of the surface  $M$ . Let's now consider a sphere of radius  $\rho$ . 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  $\tau = (\sigma_i, \sigma_j, \sigma_k)$ , where  $\sigma_i, \sigma_j, \sigma_k$  are the three edges of the triangles (and three points of  $X$ ) and are contained into the sphere or radius  $\rho$ . This triangle can be considered as the first piece of our surface and is called the seed. Finally we will note  $e_{i,j}$  the edge of the triangle linking  $\sigma_i$  and  $\sigma_j$ .

Considering these notations the algorithms works as followed:

Starting with the sphere containing the three edges of the triangle  $\tau$  we're going to make this sphere rotate around the edge  $e_{i,j}$ . If the sphere encounter another point  $\sigma_o$  then another triangle  $\tau_1 = (\sigma_i, \sigma_j, \sigma_o)$  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 tagged as boundaries. The figure below illustrate the algorithm in 2D.

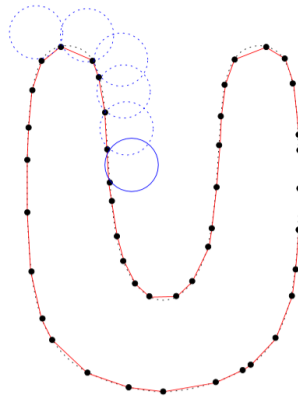


Figure 10 - Ball pivoting algorithm [10]

One of the major advantages of this algorithm is that it is really fast and robust 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:

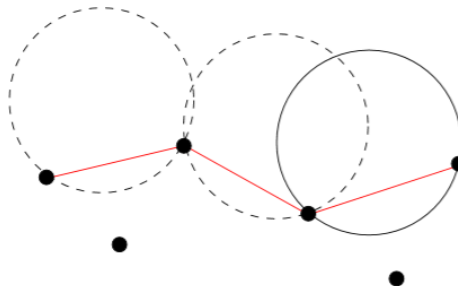


Figure 11 - Ball Pivoting algorithm in presence of noisy data [10]

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 problems if the sampled data contains missing parts. Thus the ball will "fall" through the surface when pivoting (see figure 12). However tackling this issue can be done by applying the ball pivoting algorithm several times 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  $\frac{1}{\rho}$  then all the sample points will not be taken into account (see figure 12). This is why it is extremely important to know the density of your point cloud before applying the ball pivoting algorithm.

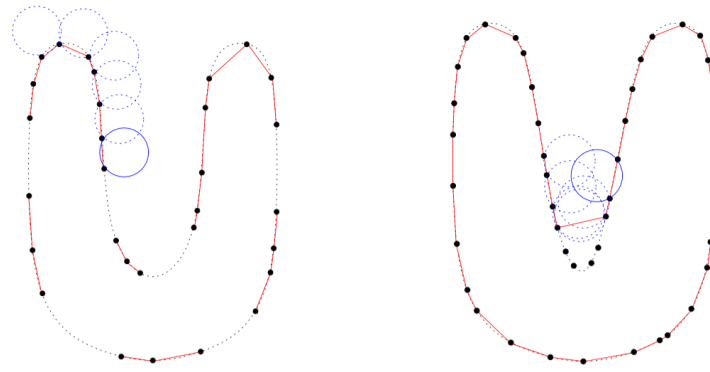


Figure 12 - Biggest issues encounter with the ball pivoting algorithm

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 don'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 decomposes the point cloud in a Voronoï diagram. This Voronoï is composed of multiples cells where a cell  $C$  associated with a vertex  $S_i$  is defined as follows:

$$C(S_i) = \{P \in \mathbb{R}^2 | d(P, S_i) < d(P, S_j) \forall i \neq j\}$$

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

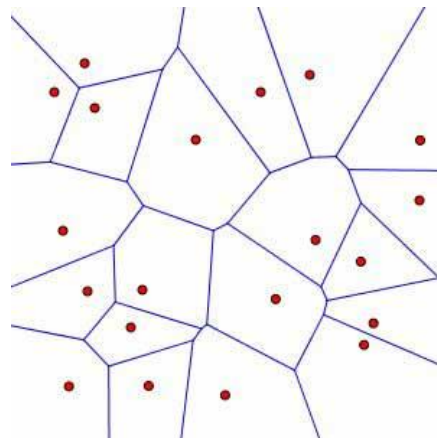


Figure 13 - Voronoï Diagram. Each red dot represents the center of the cell

We can then use the Delaunay triangulation, combined 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 points 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 many input on the 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 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 complicated and dense point cloud. The Poisson surface algorithm and ball pivoting seem to be a good way to start as they seem 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 as it is quite a big one.

Finally, another task 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 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 tour of the lighthouse. 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 possible. 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 the other rooms 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 tour 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 using a Faro laser. The specification of the laser were not known to me. The data I possessed were the combination of several scans of the room from different point of view (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 point cloud:



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. Whereas I have no idea with which methods the point clouds have been decimated or subsampled. This is an issue as I have no idea which kind of data have disappeared and also because it reduces drastically the quality of the point clouds and the quality of their textures.

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 the coordinates of each points in an easy to use list, 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.

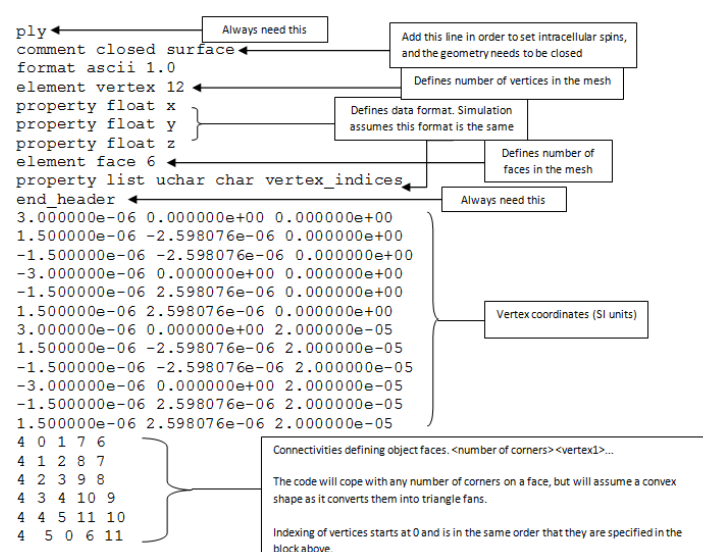


Figure 15 - Structure of a ply files were the coordinates of the vertex and their connectivities (i.e how they are linked to create faces) are stored. [12]



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 point clouds 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. I will first be working on the chamber2 as it is more challenging. However the chamber1 seems really interesting because despites its fewer number of points it has really good colors and textures.

### 5.2 Software

Before talking about the methodology itself it seems important to me to talk about the different softwares I have been using and why I chose them. The three software I have been using are Meshlab for the surface reconstruction, CloudCompare for cleaning and Blender to animate and render the final mesh. Links to written and video tutorials to learn how to use this programs are provided in the appendix 2.

#### 5.2.1 Meshlab

Meshlab is the main software I have used during my project. I chose it for many different reasons. First of all it is a totally free software. Moreover it handles really well dense point cloud 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).

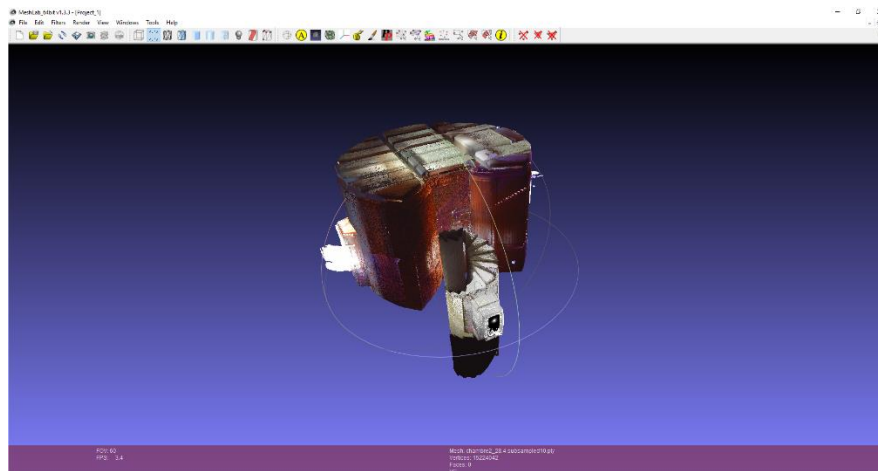


Figure 16 - The Meshlab interface

#### 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 point clouds. For example, I removed some parts and furniture I didn't want in my room as they were always giving bad results.



## UV5.4 Creation of a virtual tour from a point cloud

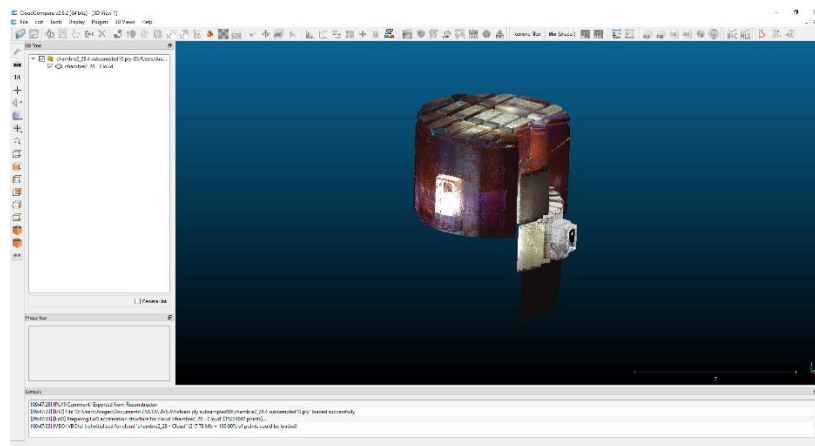


Figure 17 - The CloudCompare interface

### 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.

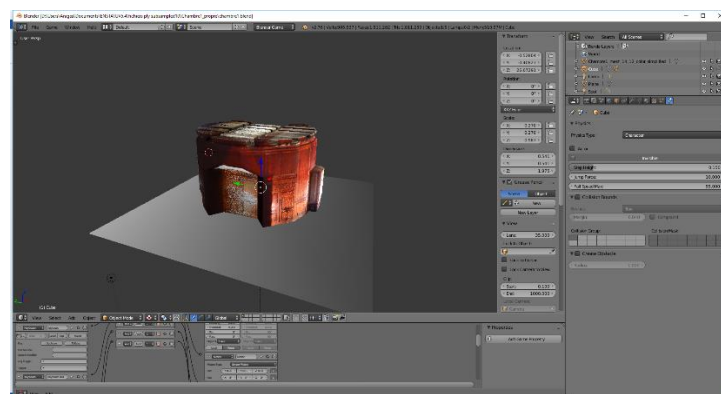


Figure 18 - The Blender interface

### 5.3 Normal computation

As I stated before in order to apply both the ball pivoting and the Poisson algorithm we first need to compute the normal for each points of the cloud. To compute the normal I have used a filter already implemented in Meshlab that estimate the normal of a point using its neighbors. Moreover this method is similar to the one explained in part 1.2. Before computing the normals I removed all the noisy points coming out of the windows of the chamber2 and chamber1 using CloudCompare to avoid problem with the normal orientation on these areas.

The filter parameters I used in Meshlab are presented on the figure 19. I chose to use 10 neighbors to compute the normal of each point as it is normally sufficient to correctly estimate the normals. Moreover estimating the quality of the normal is really hard as I have not the different position of the laser to try and correctly re-oriented the normals. I therefore had no input to add into the viewpoint fields in the normal filter.

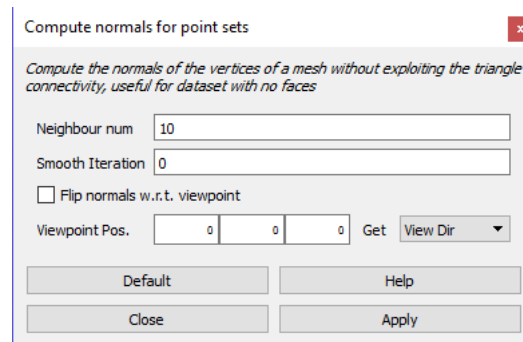


Figure 19 - Normal computation filter in Meshlab

The figure below shows the result of the filter:

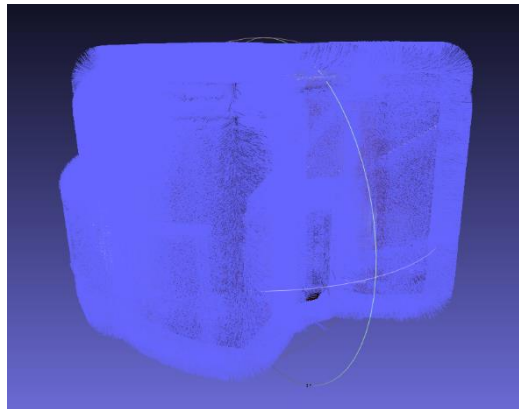


Figure 20 - Normals of the chamber2 represented by blue arrows

### 5.4 Cleaning

The second step after computing the normal is to clean the cloud from its furniture. As a matter of fact as you can see on the figure 21, where I did a quick surface reconstruction using the Poisson algorithm, the furniture in the room (table, chair and doors) are not rendering well. In my opinion this is mostly due to the fact that the implicit function near each piece of furniture is hard to compute as the point are really close and that a not well oriented normal can drastically modify the shape of the surface.

The ideal way to avoid this problem is simply to separate them of the rest of the room for reconstruction them independently. To do so I used CloudCompare to remove one by one each piece of furniture so they can be treated independently. In order to decrease the computing time I also remove the staircase that you can see on figure 17.

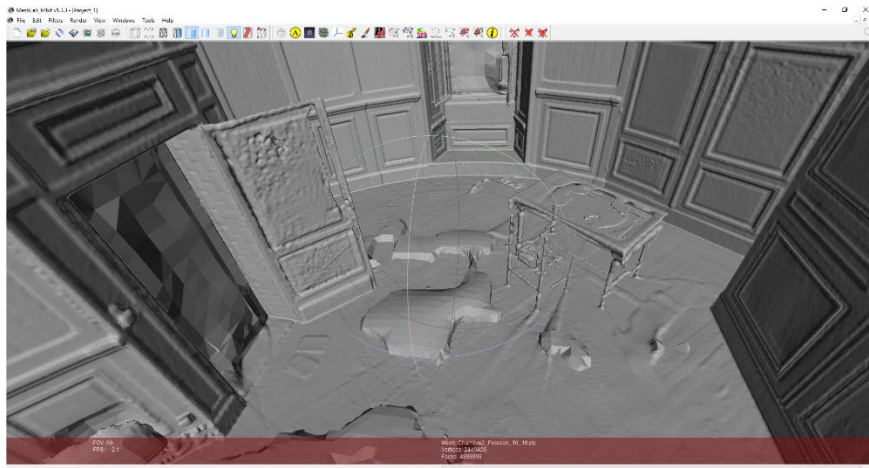


Figure 21 - Quick reconstruction of the chamber2 to see the impact of the furniture on the mesh

## 5.5 Surface Reconstruction

### 5.5.1 Poisson Algorithm

The first algorithm I tried to reconstruct the chamber2 was the Poisson one. As a matter of fact it is the most commonly used algorithm for surface reconstruction as it gives watertight meshes even with non-uniform point cloud. However as I am using Meshlab and the algorithm is already implemented I decided to try and evaluate the importance of each parameters used by Meshlab by doing several tests on the chamber 2. The results of these tests can be found in the appendix 1.

First of all let's explain below the different parameters used by Meshlab.

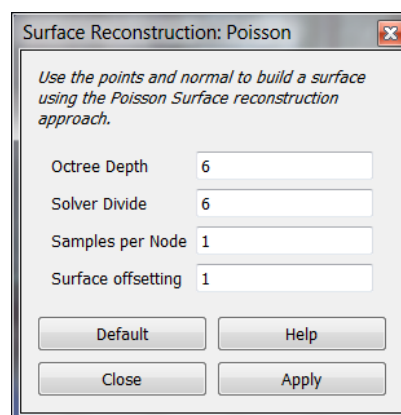


Figure 22 - Poisson reconstruction parameters in Meshlab

#### **Octree Depth: [13] [14]**

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 point cloud. To do so they divided 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.

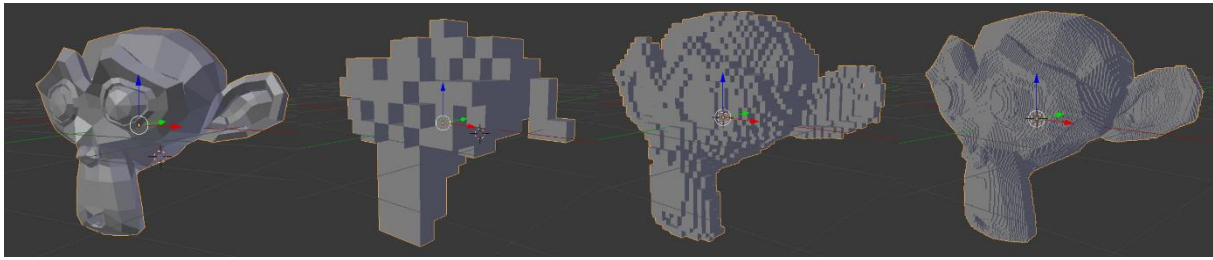


Figure 23 - From left to right: original mesh, octree depth 4, octree depth 6, octree depth 8

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 means 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  $2^d \times 2^d \times 2^d$ .

### **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 points 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. We will not try to evaluate this parameter as our surface should not have any offset.

Here the most important parameters are: The octree Depth, the Solver Divide and the Samples per Node. First of all let's talk about the octree depth. As I explained it above, the octree depth represent here the resolution on which we are going to work. For example with an octree depth of 6, we will have a spatial resolution of  $256 \times 256 \times 256$  which is not a lot for point cloud composed of a million points. The figure below illustrate the difference between a mesh created with an octree depth of 8 and one with and octree depth of 10.

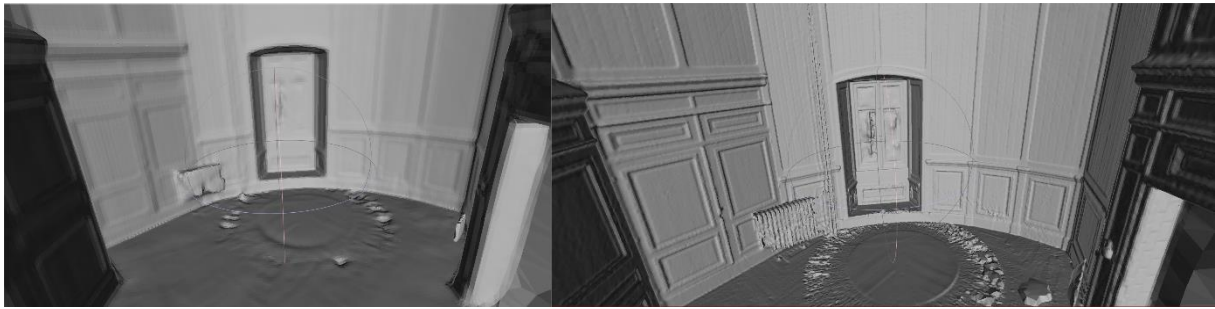


Figure 24 - On the left a reconstruction using an octree depth of 8 and on the right using an octree depth of 10

As you can see the surface using a bigger octree depth have many more details. However it here increases the noise. By doing my test I conclude that having an octree depth of 9 is highly sufficient as it provides enough details for the room to be visually really accurate, but also a number of faces relatively low so I can use the mesh in Blender. Indeed it is important to remind that I must create a virtual visit that can work on any computer. Thus it is important to find the best compromise between performance and visual quality. For example an octree depth of 12 give too many faces to be easily used in Blender or with low performance computers. I will take an octree depth of 10 for reconstructing the chamber2 as it gives slight more details than a depth of 9 but didn't add to much faces.

Concerning the solver divide it was supposed to reduce the computing time for the algorithm. However according to the result I gathered, decreasing the solver divide doesn't really improve the computation time. For example at an octree depth of 12 I only gain 6 min by using a solver divide of 8 instead of 12. Below an octree depth of 12 the time we gain by changing the solver divide is too insignificant to be taken into account.

Finally the Sample per Node parameter is the one that modify the most the computing time and the number of faces for the final mesh. As a matter of fact for an octree depth of 10 the computing time was divided by 2.5 for 15 samples per node and was divided by 3 for 20 samples per nodes. However the number of faces were also respectively divide by 2.7 and 3.4. This decreased has its advantages as is reduced the noise on the floor (cf figure 24). Whereas we also lost a lot of details. The figure below compares the room computed with an octree depth of 10 and 1 sample per node with the room computed with an octree depth of 10 and 15 samples per node.

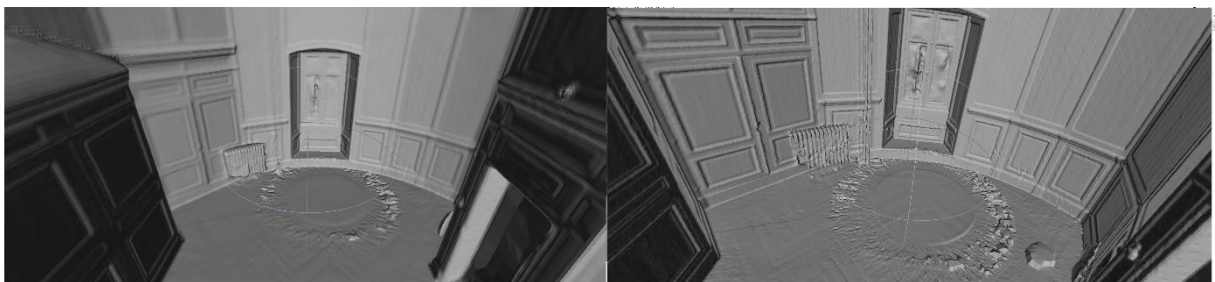


Figure 25 - On the left: 15 samples per node, On the right : 1 sample per nodes

As you can see the surface computed with 15 samples per node have less noise but still have a correct amount of details compared to 1 sample per node. It appears to me that 15 samples per node is the best choice here as it allow to keep a certain amount of details (not like 20 samples per node) and in the same time attenuate the noise.

Therefore I will use an octree depth of 10 and 15 sample per node to do the surface reconstruction of the chamber2. Concerning the chamber1 I reconstructed it using an octree depth of 14 and 1 sample per node. As a matter of fact this chamber comport less point and I therefore had to increase the



octree depth in order to get more details. As there is less points the noise is also smaller this is why only 1 sample per node is sufficient to get an accurate mesh. The figure bellow shows the two meshes I have generated.

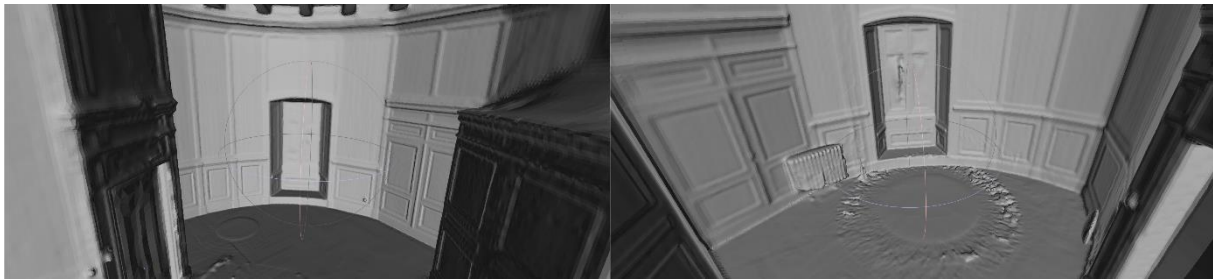


Figure 26 - On the left: the chamber1, on the right: the chamber2

As a conclusion we can say that the most important parameters are the octree depth and the samples per node (especially for a noisy point cloud). The best way to correctly choose the octree depth is to make sure that the spatial resolution of the octree is big enough to take into account as much point as possible from the point cloud. However the samples per node must be chosen empirically in my case as I have no information concerning the scans errors or that I don't have the complete cloud point. In my opinion by knowing the type of laser used and having the full scans it must be easier to correctly choose the right parameters. Testing empirically different parameters is for me the best options to create the best mesh possible.

### 5.5.2 Ball Pivoting Algorithm

The second algorithm I tried to apply on the cloud point was the ball pivoting algorithm. As the surface I created is supposedly watertight, this algorithm seemed to be a good solution as the ball creating the mesh will normally always stays inside this mesh. Let's explain the different parameters used by meshlab to compute this algorithm.

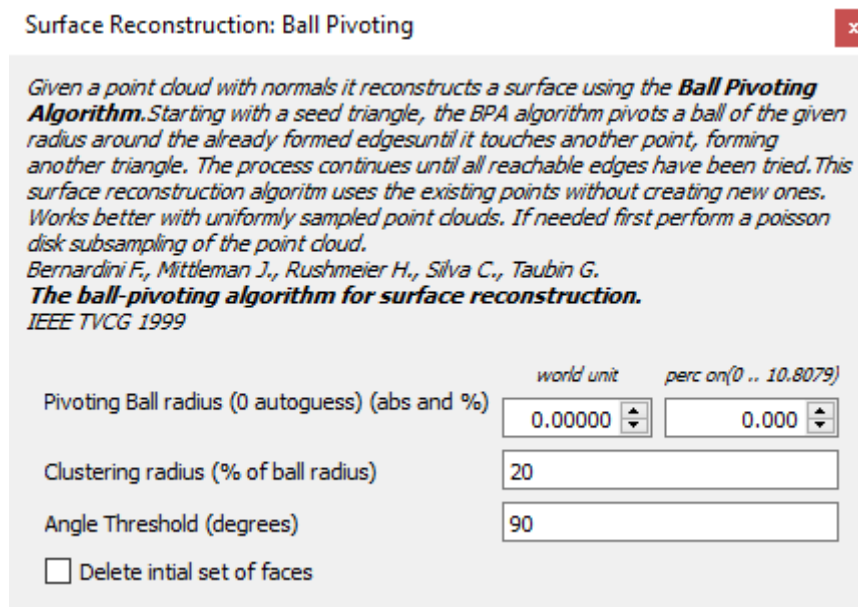


Figure 27 - Ball Pivoting parameters

#### Pivoting Ball Radius :

This parameter corresponds to the radius of the ball we will use for the algorithm

**Clustering radius:**

This parameter will merge two points if there are too close together. On the figure 27 for example, two points will be considered too close if they are within a ball radius of 20% of the ball pivoting radius.

**Angle threshold:**

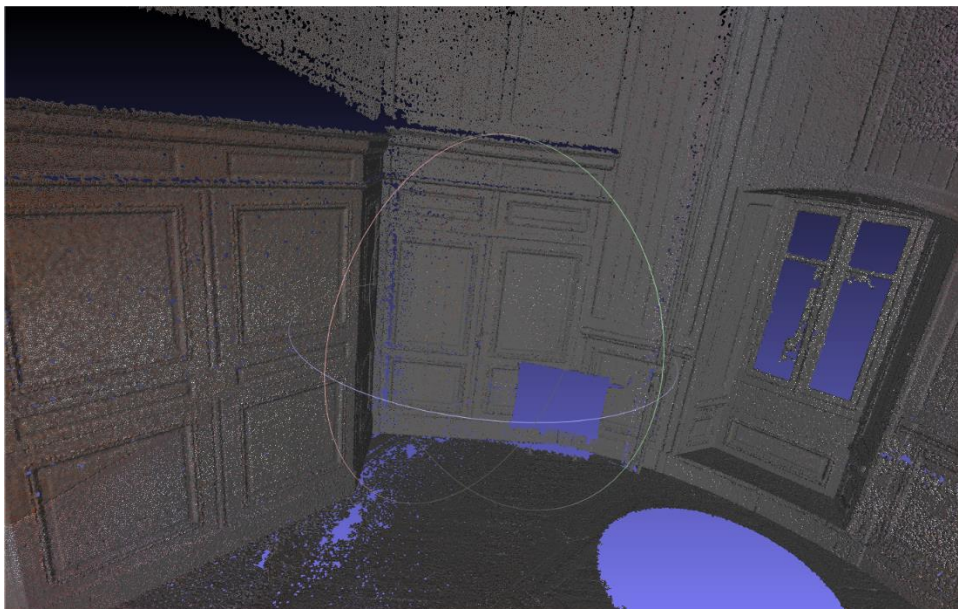
Correspond to the maximum angle at which a pivoting ball can rotate around an edge.

The most important parameter is here the radius of the ball. However choosing it, is in my case really difficult because I don't know the density of the point cloud and it appears clearly to be not homogeneous in term of vertex spacing. Moreover the radius in MeshLab is really hard to guess as the unit used to describe it is not a real unit.

I first tried to find empirically the radius on chamber2 by using the auto guess proposed by MeshLab or random radius but I never obtained satisfying result as I most of the time end up with a mesh composed of 100 faces because the ball didn't have the right radius.

This is why I have decided to use a sampling algorithm in order to get a uniform point cloud. To do so I used the Poisson Disk algorithm that is already implemented in MeshLab. This algorithm allow to sample the point cloud by only keeping points that are at a chosen distance from each other. By doing so I managed to obtain a uniform cloud point. I chose for the sampling a radius of 0.01 (in MeshLab word unit), which give a point cloud approximately 90% less dense.

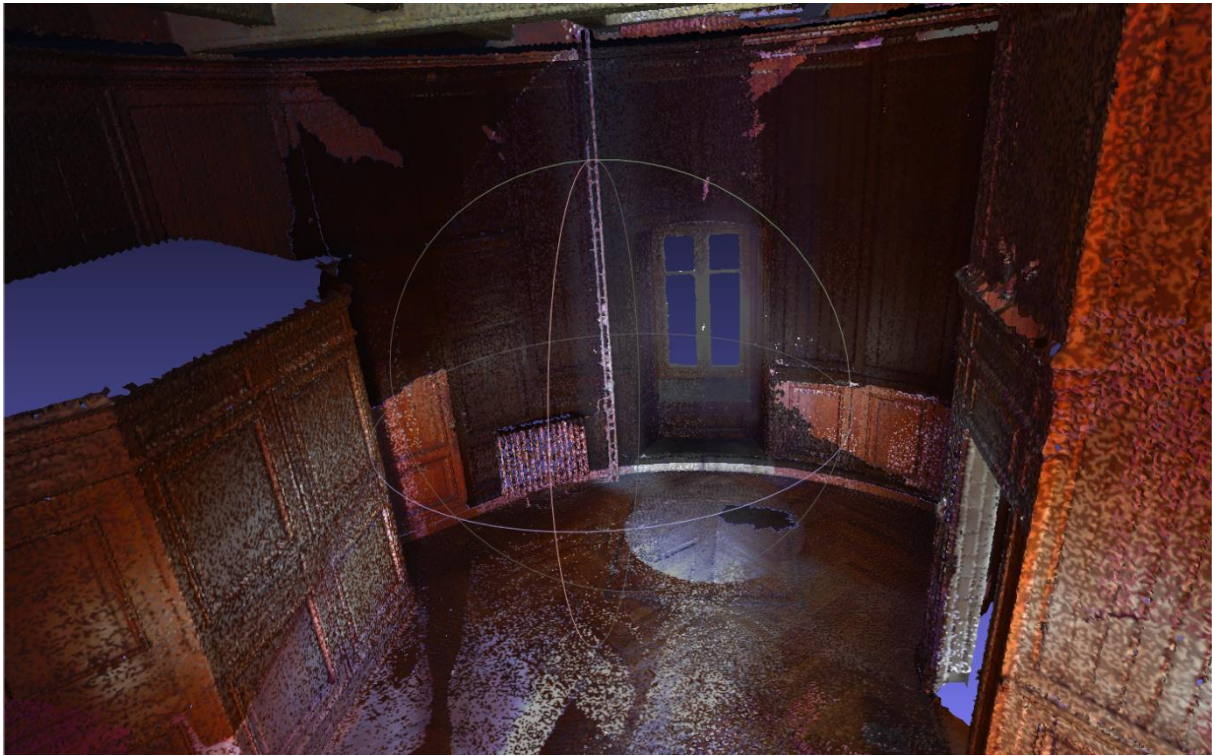
I first applied the ball pivoting algorithm on the full sampled point cloud with a ball radius of 0.011 (to be sure that she will not go through the point cloud). However even if the result was good (see figure below), it took a lot of time (around one hour and a half) to compute. Furthermore to remove the holes from the mesh I was supposed to do the algorithm many more times with other ball radius. To avoid more computing times, I've decided to decompose my point cloud into 6 different parts and to apply to each part multiple iterations of the ball pivoting algorithm with different ball radius.



*Figure 28 - The ball pivoting algorithm applied on the chamber2*

This method allowed me to quickly reconstruct each parts and adapt the number of iterations needed to correctly reconstruct the surface of each part. You can see in appendix 2 the different operations

I've done in order to reconstruct each part of the mesh. The final result (after adding each part together) can be seen on the figure below.



*Figure 29 - Result of the ball pivoting algorithm on the 6 parts of the point cloud*

As we can see the result is by far less accurate than the result obtained with the Poisson surface reconstruction. There are two major issues. The first one is that the mesh is not watertight. We can see many holes into the surface, and especially a big one (see figure 29) where the laser didn't go. The second is that the surface is much sharper i.e it is really uneven and unnatural.

As a conclusion I think that the Poisson surface reconstruction is by far a better solution for my project. It offers a better mesh, with a higher resolution and no holes and is also less tedious to do.

### 5.6 Color Generation

The next step before generating the virtual visit is to add colors to our mesh. As the .ply files that I have been given already associated with each point a color, adding it to the mesh using MeshLab is really easy. First of all, it's important to say that the mesh I have created have no color. Hopefully MeshLab implement a filter that can transpose vertex attributes, and more precisely their colors from a source point cloud (or mesh) to another mesh. This filter assumes that the source mesh and the receiving mesh are aligned and similar. Then the filters find for each source vertex the closes point on the receiving mesh (in a sphere defined by the user), and gives to it the color of the source vertex. The figure 30 shows the color on the floor of the chamber1 and chamber2.





Figure 30 - On the left: Color of chamber2, on the right: Color of chamber1

These two screenshots show perfectly the major problem with both of the chambers texture: They are not homogenous. One more time this is due to the laser scan itself. As I said before the point clouds are in my case a combination of different point cloud. The problem here is that the different scans have been done at a different hour of the day so the luminosity is not the same everywhere. A good way to tackle this issue would have been to shut the window and have a fix light during the three different scans. Another way to solve this problem may be to equalize manually the luminosity of each point, however this will not be done in this project.

As a conclusion I will keep this texture in order to get a functional virtual visit. If we really want a good texture, we can either scanned the lighthouse one more time but with a constant luminosity or we can also work with photographs that we can be applied on the 3D mesh in Blender. However we will talk about Blender in the next part.

#### 5.7. Creation of the virtual visit

The final part is now to create the virtual tour of the lighthouse. To do so I've decided to use the open source software Blender as it has a game engine that can easily be used to create the virtual tour, but is also able to create dynamic videos of 3D meshes. Moreover it is also really easy to import a .ply file in Blender and directly work on it. I decided to use the chamber1 to create the first part of the virtual tour because its textures were better than the chamber2.

As we are going to use the game engine for creating the virtual tour, we'll be facing the same problems video games developers are facing. The first one is the complexity of the object in the scene. As a matter of fact, the major work of a video game developer is to make a scene, or mesh look well, in real time. Indeed the more complex the mesh, the more power you will need to render it real time. Hopefully in our case 1 000 000 faces (which correspond to only one room) can be render and loaded by the game engine. So if we are just rendering a room at a time we will not have any problem.

Whereas we will have a problem with the complexity of the mesh when it comes to collision. Indeed when you navigate into a video game you never go through the ground or a wall because the game is constantly checking if a part of your character is not into the wall itself. However when the wall or the character is really complexed and composed of a lot of faces and vertices it can be really hard to compute in real time if there is or not a collision.

To properly tackle this issue we are going to use bounding boxes. Bounding boxes are simple shapes (most of the time cuboid) that contain a complex 3D model and allow to tell or not if there is a collision. For example on the figure below you can see that the wizard is encapsulated in a blue collision box. To avoid the wizard to enter in collision with the floor, a simplistic way to do it is to check that the four vertices of the cube are always above the vertices composing the floor.



Figure 31 - Bouding Box in a video game

In order to make the virtual tour possible I decided to model the character moving into the lighthouse as a giant cuboid, and model the floor as a simple plane. Therefore the game engine will easily be able to detect collision and avoid the character going through the floor. You can see on figure 32 the player resting on the floor I made. I also had some vertical planes to simulate the wall.

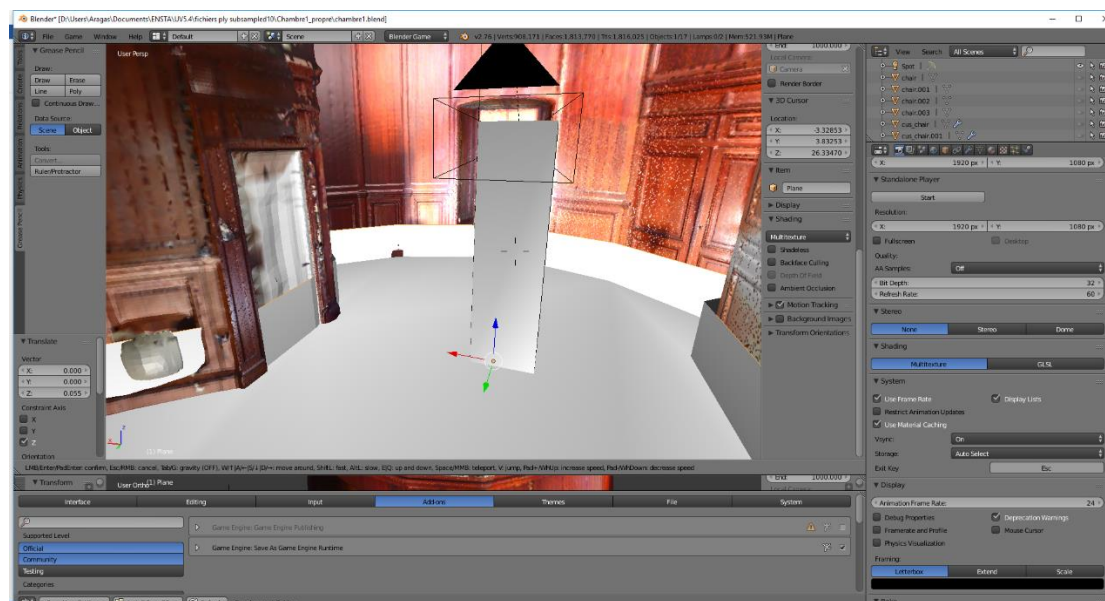


Figure 32 Player representation and bounding box for the lighthouse tour

As we want to feel like we are in the lighthouse, I added a camera on top of the cuboid (the black rectangle on figure 29). Thus, we will now be able to see through the camera.

The last step in order to make the tour possible was to allow the visitor to move and look around into the lighthouse. To do so blender used logical bricks that can be associated with a model. You can see on figure 30 that the player is associated with the arrows of the keyboard and that each arrow trigger a translation. In the same manner moving the mouse will rotate the player and the camera.

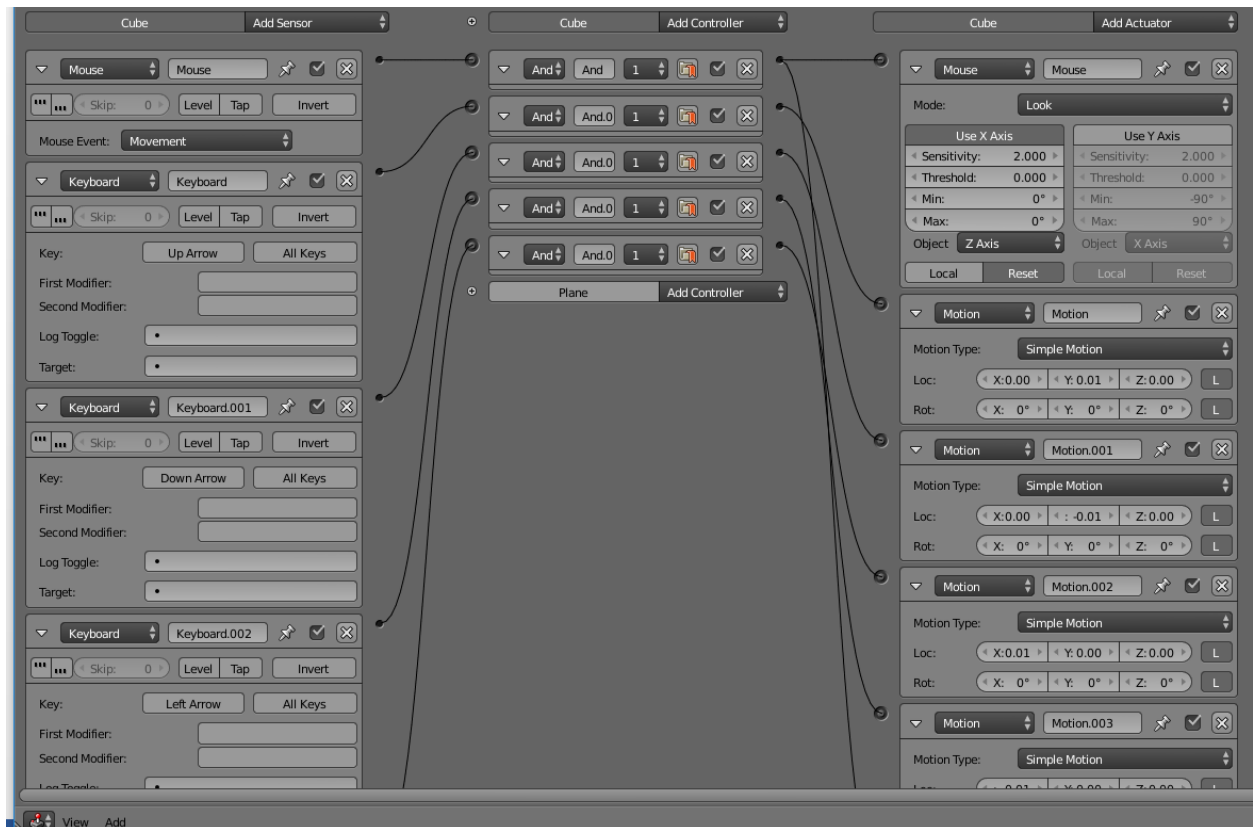


Figure 33 - Logical brick in Blender

To conclude the result of the virtual tour is really good. I created a .exe using the game blender engine that can be launched in order to move into the chamber1. The next step will be to have a more accurate mesh of the chamber and to do the same work for the rest of the lighthouse.

## 6 Improvements and limitations

The creation of the prototype for the virtual tour have been a real challenge as I face many different issues and the result can be greatly improve at each stage of the process.

First of all one of the best way to improve this 3D representation is to work directly on the raw data instead of using pre-treated point cloud. As a matter of fact the raw data were probably denser than subsampled point cloud I worked on and therefore would have provided more information.

Another way to improve the result is to know more about the scans, and more especially the position of the laser for each scan, and which are associated with which scan. These information would have allow to obtain much more accurate normals for the model, thus a better mesh. Furthermore knowing the specification of the laser would have led to a better understanding of the point cloud and especially its density.

Concerning the textures of the mesh a good idea would be to have pictures of the lighthouse that can be directly remap on the mesh using Blender. This will drastically improves the final result by adding a touch of realism.

Moreover doing a virtual tour of the lighthouse in its entirety is another challenge as it will involve to work on much bigger files and data. For example the blender file I generated for the virtual tour weight 165 Mo which is already really big for only one room. It is imperative to find a way to reduce the size of the files.

## 7 Conclusion

As a conclusion I can say that the surface reconstruction from a point cloud is really a dense and complexed subject. As we saw into the first part many papers already work on the subject, and offer to me really good basis to understand the challenges I will be facing.

Using this basis I managed to describe in this report one methodology to reconstruct a watertight indoors environment from point cloud I know nearly nothing of. If this lack of knowledge around the data were at first a problem, it also push me to find the right tools, softwares and methods to correct my work.

However, it is important to keep in my mind that point clouds can be really complex and different, and will bring different problems with them. Automating the process is therefore really difficult to do as a point cloud is an implicit representation of an object or scene we have scanned. Automatization is for me the next step into surface reconstruction.

## APPENDIX 1: Poisson Parameters (filter by number of faces)

Octree Depth	Solver Divide	Sample Per Node	Time (ms)	Number faces	Number Vertices
6	6	15	14451	8849	17694
6	6	5	14594	8849	17694
6	6	1	14653	8849	17694
8	8	15	40578	299188	149602
8	6	15	29983	299224	149620
8	8	20	40504	300272	150142
8	6	1	40008	301596	150800
8	8	5	39033	301608	150798
8	8	1	38626	301630	150817
8	7	1	38379	301676	150840
9	9	15	119251	1096702	548381
9	9	1	112194	1218094	609057
9	8	1	114934	1218104	609062
10	9	20	159386	1391270	695647
10	8	20	158689	1391272	695648
10	10	20	156289	1391272	695648
10	7	20	161230	1391296	695660
10	6	20	174373	1391386	695697
12	12	20	187879	1554954	777496
12	8	20	191239	1554974	777506
10	7	15	194952	1731706	865871
10	9	15	191774	1731730	865883
10	8	15	191791	1731766	865901
10	6	15	212402	1731940	865986
12	9	15	233284	1979940	989981
12	10	15	227745	1979944	989983
12	12	15	225969	1979950	989986
12	8	15	232584	1980032	990027
14	8	15	1019280	2003036	1001535
10	10	5	374005	3600144	1800068
10	7	1	485257	4731440	2365707
10	8	1	478991	4731618	2365798
10	9	1	476899	4731670	2365826
10	10	1	477715	4731776	2365877
10	6	1	529643	4732004	2365993
12	12	1	1923488	14891740	7445669
12	9	1	1561219	14891750	7445673
12	10	1	1558248	14891782	7445690
12	8	1	1585258	14891928	7445766

## APPENDIX 2 : Ball Pivoting Algorithm

### Ball Pivoting

Centre_gauche	Radius	Time
	0,011	38290
	0,015	19820
	0,02	2775
	0,025	2499
	0,1	2399
	0,2	2166

Centre_droit	Radius	time	Number Of faces
	0,011	12593	154299
	0,015	4445	16143
	0,016	1197	1167
	0,017	1145	350
	0,018	1119	167

Droite	Radius	time	Number Of faces
	0,011	50724	340599
	0,012	4949	579
	0,014	14621	40086
	0,2	7925	16138

Gauche	Radius	time	Number Of faces
	0,011	36209	391040
	0,012	6630	11878
	0,013	3594	3594
	0,018	3497	4754

#### UV5.4 Creation of a virtual tour from a point cloud

0,02	3130	978
0,035	2990	402

Sol	Radius	time	Number Of faces
	0,011	27821	331573
	0,012	5719	11442
	0,013	3281	6018
	0,014	2825	4429
	0,015	2752	2799
	0,016	2557	2024
	0,017	2443	1375
	0,018	2448	924
	0,019	2278	593

Toit	Radius	time	Number Of faces
	0,011	107415	424172
	0,012	250353	31586
	0,013	15001	26685
	0,014	9946	13143
	0,015	8331	11484
	0,016	13281	6492
	0,017	5827	3126
	0,018	5326	2119
	0,019	4925	1323
	0,02	4684	905



## APPENDIX 3 : Tutorials

### Meshlab :

Link to the youtube channel of Mister P. This channel describe really well all the basics you'll have to know in order to manipulate correctly meshlab from point selection to applying filter.

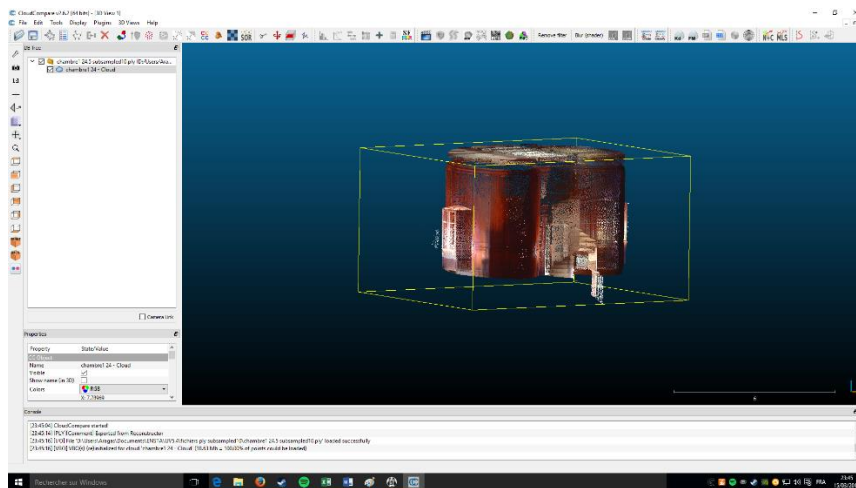
[https://www.youtube.com/channel/UC70CKZQPj\\_ZAJ00srm6TyTg](https://www.youtube.com/channel/UC70CKZQPj_ZAJ00srm6TyTg)


The presentation below explain also everything a new user have to know in order to correctly use meshlab

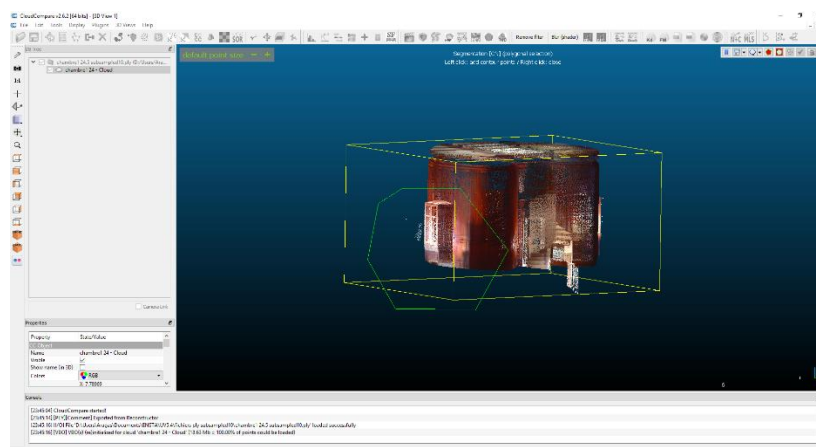
<http://www.cse.iitd.ac.in/~mcs112609/Meshlab%20Tutorial.pdf>

### CloudCompare: Cleaning a point cloud




- First open you point cloud into cloud compare and select it in the left window. If it is correctly selected most of the icons in the top toolbar should be colorful just like on the screenshot below

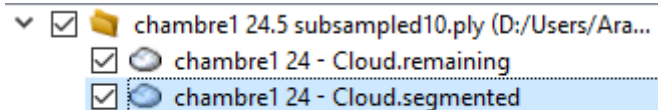


- In order to remove unwanted point choose the segment option  in the toolbar. The software should now look like this:





- You can now select the part you want to remove using left click (can be seen on the screenshot above). When you have selected the part you want to remove press right click to validate the selection. You can now divide your cloud point using the segment out button 
- If you want to remove another part of the point cloud you can do it by pressing the pause segmentation button  and re-do the third last step.
- When you have completely cleaned your mesh simply press the confirm segmentation button 
- You should now have two different point cloud displayed into the left window.



### Blender: BlenderGame Engine

The video tutorial below gives all the basics you need in order to create a first person game using blender:

Part 1: <https://www.youtube.com/watch?v=YFzLWXwfPbg>

Part 2: <https://www.youtube.com/watch?v=ydya0A3cyRk>

You can also check these links for further information:

<https://openclassrooms.com/courses/debutez-dans-la-3d-avec-blender>

<https://openclassrooms.com/courses/debutez-dans-la-3d-avec-blender/introduction-au-game-blender>

## 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](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](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].
- [13] « Meshing - Octree, Degree & Samples p... », *Matter And Form*. [En ligne]. Disponible sur: <https://matterandform.desk.com/customer/en/portal/articles/2107547-meshing---octree-degree-samples-per-node-explained>. [Consulté le: 14-mars-2016].
- [14] « GPU Gems - Chapter 37. Octree Textures on the GPU ». [En ligne]. Disponible sur: [http://http.developer.nvidia.com/GPUGems2/gpugems2\\_chapter37.html](http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter37.html). [Consulté le: 10-mars-2016].

