3D Scanner (V2)

Software Engineering Project

Under the guidance of

Prof. Yohan Fougerolle

Mr. Cansen Jiang
Mr. David Strubel

ASRAF ALI Abdul Salam Rasmi
SULAIMAN Jamilu
Masters in Computer Vision
Universit de Bourgogne
France

January 2018

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

A 3D scanner is a device that analyses a real-world object or environment to collect data on its shape and possibly its appearance such as colour. The "digital information" captured by a 3D scanner is called a point cloud. Each point represents one measurement in space. To connect these points, algorithms like Iterative Closest Point (ICP) can be used to get the point-space information and to create polygonal meshes using regular or irregular polygons, such as triangles, quadrilateral and so on. The Surfaces can be constructed on the polygon model or its shape can be used to construct digital three-dimensional models. The process of capturing digital information about the shape of an object with equipment (e.g kinect) to measure the distance between the scanner and the object is called 3D scanning. 3D scanning can capture data of very small up to very big objects or environments. 3D scanning has wide area of applications, which includes computer aided-inspection, reverse engineering, games, virtual reality, movies, medicine, marine, architecture, historical artifact, education and so on [1].

1.2 Objectives

The objective of this project is to study and modify the software engineering project titled 3D SCANNER which submitted to Centre Universitaire Condorcet Le

Creusot, University of Bugundy, France, on January 9, 2017 by students of software engineering project Group four.
Our final goals are:

- to make the codes even short and precise,
- to use a different GUI interface Window,
- to separate the codes of the project into various classes in order to make it more clear and open for future upgrades as in reuse concept.

[2]

Chapter 2

Background

2.1 Kinect

Kinect is a Light Scanner that produces real time depth maps. It enables easy access to acquire colored 3D point clouds (also known as RGB-D image) from which information such as colour and depth of each pixels can be extracted [3]. The Kinect for Windows SDK, created by Microsoft Research, is a programming toolkit for application developers. It includes drivers, APIs for raw sensor streams and natural user interfaces, installation documents, and resource materials. It provided Kinect capabilities to developers who build applications with C++ or Visual Basic by using Microsoft Visual Studio.

Figure 2.1: Kinect

The main elements of Kinect are IR light emitter, Monochrome CMOS sensor (Depth Sensor), RGB camera (Color Sensor). The Kinect converts the patterns captured by the CMOS sensor into depth maps [4]. The depth maps matched with its respective color pictures captured by the color sensor and they are combined to form a single RGB-D image that is fed to the program.

2.2 3D Reconstruction

3D reconstruction is the technique used in 3D Scanners to build a complete 3D models acquired from a set of 2D images. In this project, Kinect Sensor was used for reconstruction of 3D model. Kinect sensor can provide the distance between the sensor and each points of the object in terms of depths.

The major process carried out in 3D Reconstruction are as follows:

- **Surface Measurements:** The kinect provides us the depth of each point of the object surface in the form of 2D images.
- **Sensor Pose Estimation:** Iterative Closest Point Algorithm is employed to minimize the difference between the present and previous frame followed by applying transform matrix to obtain global coordinate system.
- **Filtering:** Pass Through Filter and Statistical Outliers Removal Filter are used to remove noise from the Point Cloud.
- **Reconstruction:** The surface is extracted from the global model.

Chapter 3

Software Requirements

3.1 Latex

LateX is basically a typing system that includes features for the preparation of Professional documents like Scientific Articles, Presentations and Presentations and more other important documentation [5]. It is a high quality typesetting system that encourages the authors to be more specific on the contents rather than the appearance of the documents. This report was written in LateX which gave us many advantages for instance writing the mathematical equations in LateX is much easier than writing it in other tools like Microsoft office word.

3.2 UML

Unified Modelling Language (UML) provides the user a standard way to predict structure of the system, known as the Class diagram. A class diagram helps the user to understand the relationship between various classes of a program [6]. A class diagram can be represented with a rectangle box with three compartments: the uppermost contains the name of the class, the middle one contains the class attributes and the last one contains the class methods. Class members (attributes and methods) have a specific visibility assigned to them, for instance, See table below for how to represent them in UML.To denote the relationship between the classes different arrows are used as shown in the Figure below.

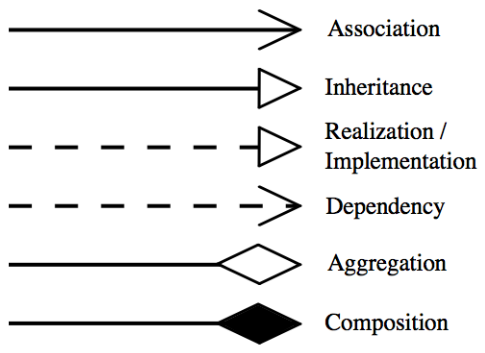

Figure 3.1: Arrows showing various relationship between the classes

3.3 Git

Git is basically a control system useful in managing and sharing collaborative works, especially in the field of programming. Git comes with an advantage of tracking the history of entire changes made to the codes which makes it more secured. It is very optimized as it allows to commit changes, create branches, merge codes, and compares the codes with the previous section. It supports a variety of nonlinear development work flows, henceforth it is more flexible.

Github is a web based Git repository hosting service. In this project GitHub was used to store all the data in a cloud which can be easily accessed by any developer. GitHub holds all data in Git format from where it can be extracted with Git tools. This made it an effective means of communication for this project and provides the possibility of collaborative work.

3.4 QT Creator and QT Framework

QT is a cross-platform software development tool which can run on various software and hardware configuration. It can run with GUI for various applications in embedded, computer and mobile platforms. Comparing to other Integrated Development Environment (IDEs) QT has full features of C++, supports Object oriented Programming, has a simplified GUI editor and above all it is compatible with all the operating systems.

3.5 Microsoft SDK

A Software Development Kit (SDK) is a combination o tools used to develop software which are specific or an operating system. Microsoft Windows SDK from Microsoft Corporation provides the developer, tools, documentation, header files, libraries, code samples and compilers which are useful in creating an application. Microsoft SDK can be integrated with Kinect sensor to develop applications that support gesture and voice recognition.

3.6 OpenNI

OpenNI is an Application Programming Interface (API) to access Prime Sense Compatible Depth Sensors. It bridges an interface between the sensors and OpenNI recording created with depth sensors. It allows an application to initialize a sensor and receive depth, RGB, and IR video streams from the device.

OpenNI comes with an uniform interfacing that allows third party middle ware developers to interface with depth sensors. Hence, Applications can make use of both the third party middle ware as well as underlying basic depth and video data provided directly by OpenNI source.

3.7 Point Cloud Library

The Point Cloud Library (PCL) is an open source project library written in C++ to process point clouds (in this case for reconstruction). It is a cross platform library which can be deployed in all operating systems [7].

PCL Framework consists of numerous algorithms for the processing of Point Clouds which includes filtering, feature estimation, surface reconstruction, registration,model fitting and segmentation. PCL was adopted in this project due to the facts that it provides all required functionality right from the acquisition of data to the surface creation and also it is free for commercial and research use [2].

3.8 Kinect Fusion

Microsoft Kinect Fusion is a tool for reconstructing a 3D model of an object using the data acquired fro the Kinect Sensor. It allows the user to interact with the 3D model of the scene.

Chapter 4

Project Management

4.1 Task Segregation

Initially the task was divided into four among persons of the group.
The four tasks are as follows:

- **Task 1:** Analyzing the old codes detecting the errors and implementation.
- **Task 2:** Segregating codes into different classes.
- **Task 3:** Learning and understanding the concepts of PCL and Open CV.
- **Task 4:** Documentation and Resource Management.

Each task was assigned to one person of the group because initially we were four in number but two left the course for some reasons so then we both decided to work together. We realized that we are late to take the concept of Open CV and we decided to stick to the segregation and enrichment of the codes. Additionally working with the GUI was the next challenging task we took into account.

4.2 Weekly Reports

We are supposed to update our work every week (what we have done in the week) into the GitHub. Due to inefficient time management and too much work load we couldn't do it update the work for every week but we managed to complete the project on time.

4.3 Coding Standards

The following coding standards are followed as much as possible in this project.

- English should be the only language used for naming any entity or for commenting.
- Function names, variable names, and file names should be as descriptive as possible.
- Variable names and types should be nouns.
- Function names usually should be phrased as an imperative
- Variable names should begin with a lower case letter.
- Words embedded in variable names begin with either an upper case letter or an underscore.
- Constants names should be all upper case and words should be separated by underscore.
- Functions should start with a capital letter and should have a capital letter for each new word. No underscores(_) should be used.
- Accessors and mutators (get and set functions) should match the name of the variable they are getting and setting.
- Types (classes, structs, typedefs, and enums) begin with an upper case letter and have a capital letter for each new word. No underscores should be used.
- File names should be all lowercase and can include underscores. C++ files should end in .cpp and header files should end in .h.
- Variables representing GUI components should preferably be suffixed by the component type name. Examples : MainWindow.
- The plural form should preferably be used for names representing a collection of objects.

- The prefix is should preferably be used for Boolean variables and methods.

Chapter 5

Implementation

5.1 Development Overview

This chapter presents information about the development of 3D scanner software, so called 3D-Scanner(V2). As explained in Chapter 2, Qt Creator is the IDE [8] used to develop this project.

Besides the project explanation, a comparison between the current project and the last year project (3D SCANNER) has been made, to explain the changes and improvements. To start with, we divided the codes into different classes, where each class holds in it, a set of functions.

As shown in the Figure5.1, the program is divided into three different classes.

- QMainWindow
- k2class
- Point2Mesh

5.2 MainWindow class

The **MainWindow** class represents the GUI Window. It is the one responsible to control the interaction between the user and the Graphic User Interface (GUI). To achieve this interaction, this class creates two instances from the other

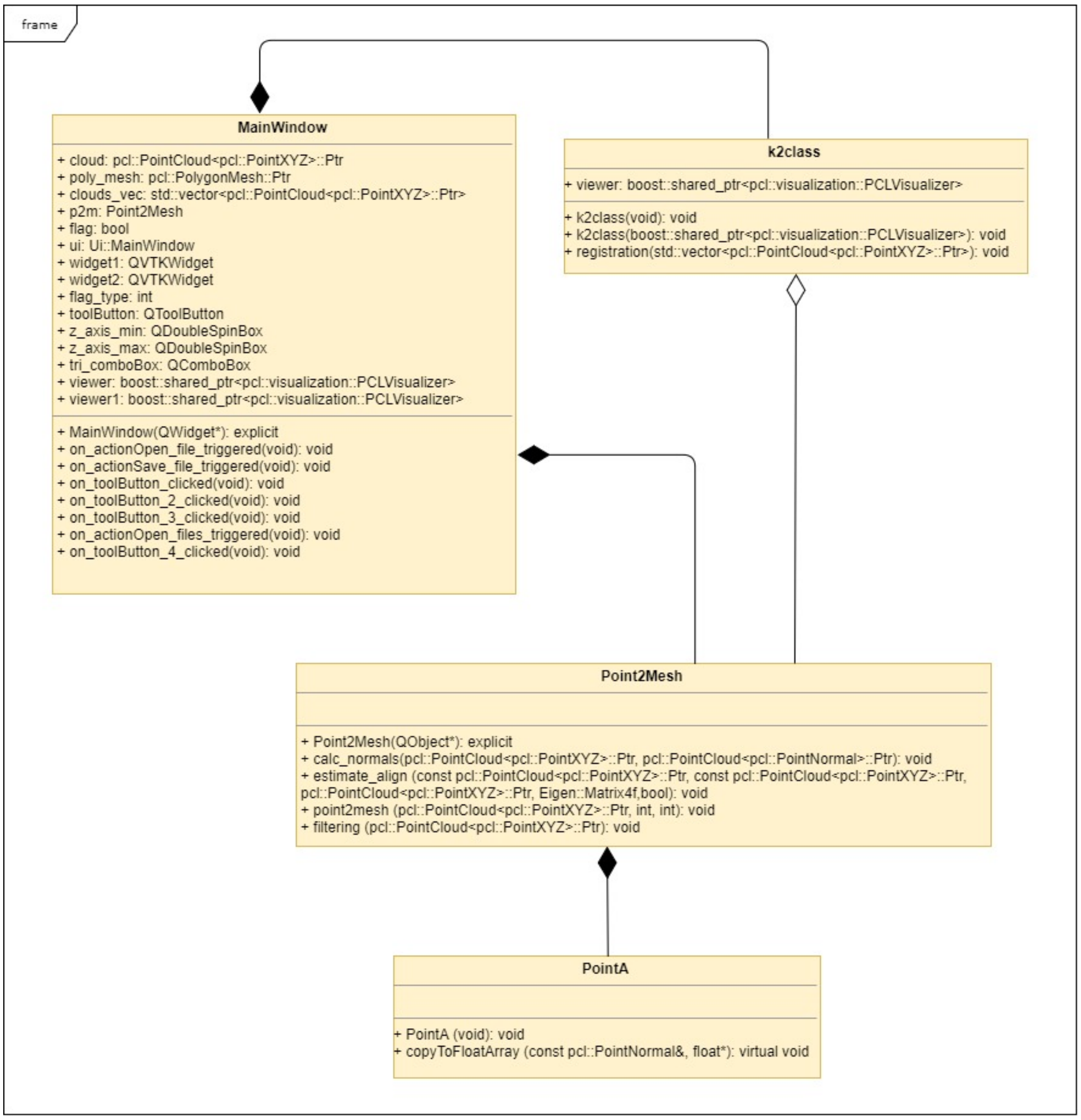
frame
MainWindow
+ cloud: pcl::PointCloud<pcl::PointXYZ>::Ptr
+ poly_mesh: pcl::PolygonMesh::Ptr
+ clouds_vec: std::vector<pcl::PointCloud<pcl::PointXYZ>::Ptr>
+ p2m: Point2Mesh
+ flag: bool
+ ui: Ui::MainWindow
+ widget1: QVTKWidget
+ widget2: QVTKWidget
+ flag_type: int
+ toolButton: QToolButton
+ z_axis_min: QDoubleSpinBox
+ z_axis_max: QDoubleSpinBox
+ tri_comboBox: QComboBox
+ viewer: boost::shared_ptr<pcl::visualization::PCLVisualizer>
+ viewer1: boost::shared_ptr<pcl::visualization::PCLVisualizer>
+ MainWindow(QWidget*): explicit
+ on_actionOpen_file_triggered(void): void
+ on_actionSave_file_triggered(void): void
+ on_toolButton_clicked(void): void
+ on_toolButton_2_clicked(void): void
+ on_toolButton_3_clicked(void): void
+ on_actionOpen_files_triggered(void): void
+ on_toolButton_4_clicked(void): void
k2class
+ viewer: boost::shared_ptr<pcl::visualization::PCLVisualizer>
+ k2class(void): void
+ k2class(boost::shared_ptr<pcl::visualization::PCLVisualizer>): void
+ registration(std::vector<pcl::PointCloud<pcl::PointXYZ>::Ptr>): void
Point2Mesh
+ Point2Mesh(QObject*): explicit
+ calc_normals(pcl::PointCloud<pcl::PointXYZ>::Ptr, pcl::PointCloud<pcl::PointNormal>::Ptr): void
+ estimate_align (const pcl::PointCloud<pcl::PointXYZ>::Ptr, const pcl::PointCloud<pcl::PointXYZ>::Ptr, pcl::PointCloud<pcl::PointXYZ>::Ptr, Eigen::Matrix4f,bool): void
+ point2mesh (pcl::PointCloud<pcl::PointXYZ>::Ptr, int, int): void
+ filtering (pcl::PointCloud<pcl::PointXYZ>::Ptr): void
PointA
+ PointA (void): void
+ copyToFloatArray (const pcl::PointNormal&, float*): virtual void

Figure 5.1: Class Diagram

classes: **k2class** and **Point2Mesh**. Thus, once the user presses a button, or choose an option on the menu bar, these parameters will be passed into those classes for 3D scanner and mesh generation, respectively. Figure 5.2 shows the GUI developed for this project.

To create GUI, Qt Designer [8] was used. This software comes with the Qt Creator IDE and allows the creation of User Interface (UI) tools for embedded devices and applications. Qt Designer provides a whole set of tools that the developer can just drag and drop into their interface, generating an eXtensible Markup Lan-

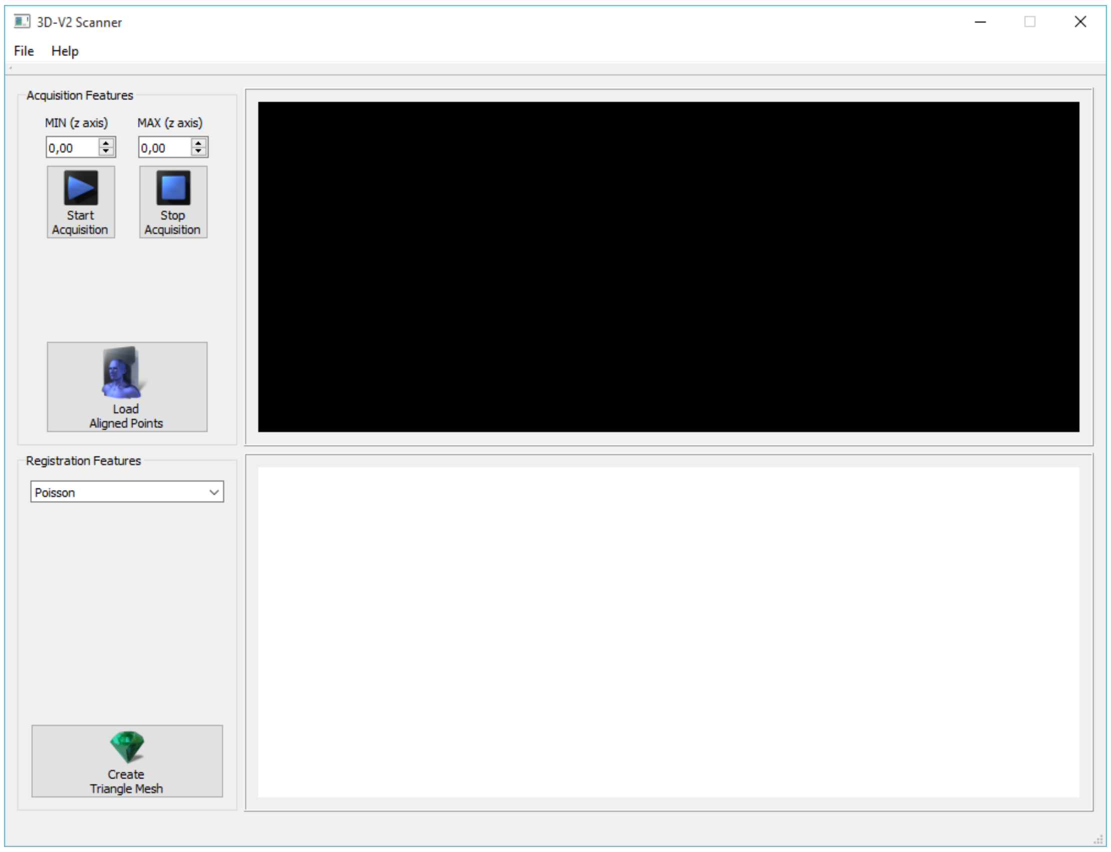

Figure 5.2: Screenshot from developed GUI

guage (XML) file. Thus, for our project, we created a file named **mainwindow.ui**, that handles all the user interaction as explained above.

5.2.1 Comparison between two projects

The 3D SCANNER project interface was built using the parent Qt class: QMainWindow, same as the one developed in this project. The main difference is that they divided the window into QFrame widgets (Figure 5.5), and for each stage of the processing part, they were setting which QFrame should be visible or not, using the **setVisible(bool)** function.

5.3 Point2Mesh class

The next class in Figure 5.1 is the **Point2Mesh**. This class was developed to handle all point clouds and polygon meshes functions, thus, in the future, it will be

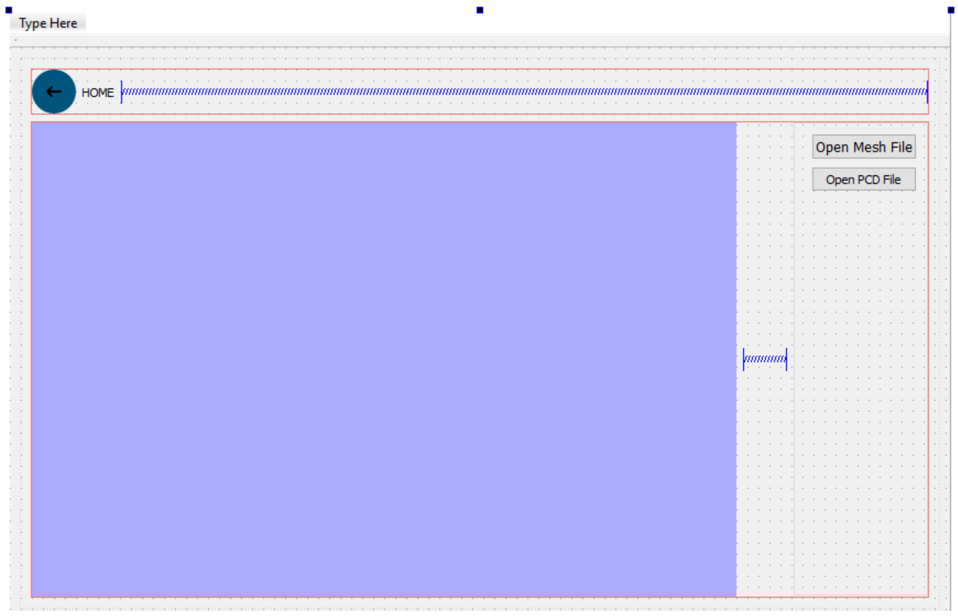

Figure 5.3: Screenshot from developed 3D SCANNER GUI

possible to use another scanner device and just pass the point cloud scanned into **Point2Mesh**. This class is composed by four methods:

- calc_normals
- estimate_align
- point2mesh
- filtering

The **calc_normals** function, receives as parameters: the point cloud scanned and the point cloud to normal pointers, that will be filled into the function. For normal calculation, a PCL [2] function called **Normal Estimation** was used as explained in Algorithm 1 extracted from PCL website [2].

Algorithm 1: Normal estimation algorithm from PCL [2]

```
Result: Normal Point Cloud
1 for each $p \in$ cloud do
2     get the nearest neighbors of p;
3     compute the surface normal n of p;
4     check if n is consistently oriented towards the viewpoint and flip otherwise;
5 end
6
```

The next function, called **estimate_align** was developed to execute the alignment between all the captured point clouds from Kinect V2. This concept is well know as registration and computes the set of point clouds in pair (two-by-two). Thus, our developed function receives two point clouds (**cloud1** and **cloud2**), an pointer for the output point cloud (cloud_final), a transformation matrix (**transf_m**), provided by Eigen library and a Boolean flag (**flag**) to compute the reduction of samples.

5.3.1 Sensor Pose Estimation(ICP Algorithm)

The main idea of this function is the use of a non-linear Iterative Closest Point algorithm. The ICP algorithm is used for geometric alignment of 3D models 3 when an initial estimate of the relative pose is known. In the ICP algorithm, one point cloud, the reference, is kept fixed, while the other one, the source, is transformed to best match the reference. The algorithm iteratively revises the transformation needed to minimize the distance from the source to the reference point cloud.

After scanning a person, the next task is to combine the consecutive scans into a single point cloud. To achieve that, the first frame is fixed as reference of the coordinate system to be used. All the other frames will be transformed into that coordinate system. This transformation can be calculated iteratively using the ICP algorithm which will give as a result the alignment of the successive individual scans of point clouds into a unified 3D global map.

5.3.2 Filtering

After the alignment step, the point cloud can be passed through a filter to reduce noise and estimate a better number of points for representation. This filter is provided by the developed function called **filtering**, that receives as parameter only the original point cloud and using the Statistical Outlier Removal algorithm provided by PCL [2], reduces the noise by the calculation of point distribution by their neighborhood [2] based on Gaussian function of mean and standard deviation, which values are provided into the function, and in our case, it was set up as 10 and 1.0, respectively.

5.3.3 Reconstruction

The last step for 3D reconstruction is composed by triangulation algorithms.

Greedy Triangulation (GT)

Greedy Triangulation (GT) is an algorithm that receives a point of clouds and converts it into a triangle mesh by projecting the local neighborhoods of a point along the points normal and connecting unconnected points. We perform GT algorithm to create a triangular mesh from the smoothed point of clouds.

The focus of GT is to keep a list of possible points that can be connected to create a mesh. Greedy Triangulation is obtained by adding short compatible edges between points of the cloud, where these edges cannot cross previously formed edges between other points Greedy Triangulation can, however, over-represent a mesh with triangles when these are represented by planar points. GT algorithm contains accurate and computationally efficient triangulation when the points are planar. However, a big difference can be noticed when the triangulation is non-planar, this can be fixed by increasing the number of boundary vertices or linearly interpolating between the boundary vertices.

An approach at computing the Greedy Triangulation is to compute all distances, sort them and examine each pair of points in length and compatibility with edges created. The Greedy Triangulation algorithm is simple at its core but also not very reliable. With a point cloud, S, of n points, the algorithm looks for the closest point where a compatible edge can be created between the two. A compatible edge can be described as an edge between two points that does not intersect with any other edge. In the Point Cloud Library, greedy projection triangulation works locally, however it allows the specification of the required features to be focused on. Such parameters are neighborhood size of the search, the search radius and the angle between surfaces.

Poisson Triangulation

Poisson is a mesh reconstruction from oriented points which can be casted as a spatial Poisson problem. We perform the Poisson algorithm to obtain a watertight mesh from a set of smoothed and filtered point of clouds. The Poisson reconstruction algorithm reconstructs a watertight model by using the indicator function and extracting the iso-surface. The indicator function is a constant function, obtained from the sampled data points, whose computation would result in a vector with unbounded values.

The surface integral of the normal field is approximated using a summation of the point samples and then reconstructed the indicator function using its gradient field as a Poisson problem. The gradient field is used to reconstruct the indicator function of the surface.

In this project, the triangulation function, called as **point2mesh**, was based on two methods, allowing different visual approach for the user. The function receives as parameter the composed and filtered point cloud (**cloud_pcl**) from the two methods described above and an integer (**type**), representing the triangulation type, where 0 is for Poison and 1 is for Greedy approach.

Thus, into our GUI, they can choose between Poisson Triangulation (Figure 5.4) or Greedy Triangulation (Figure 5.5).

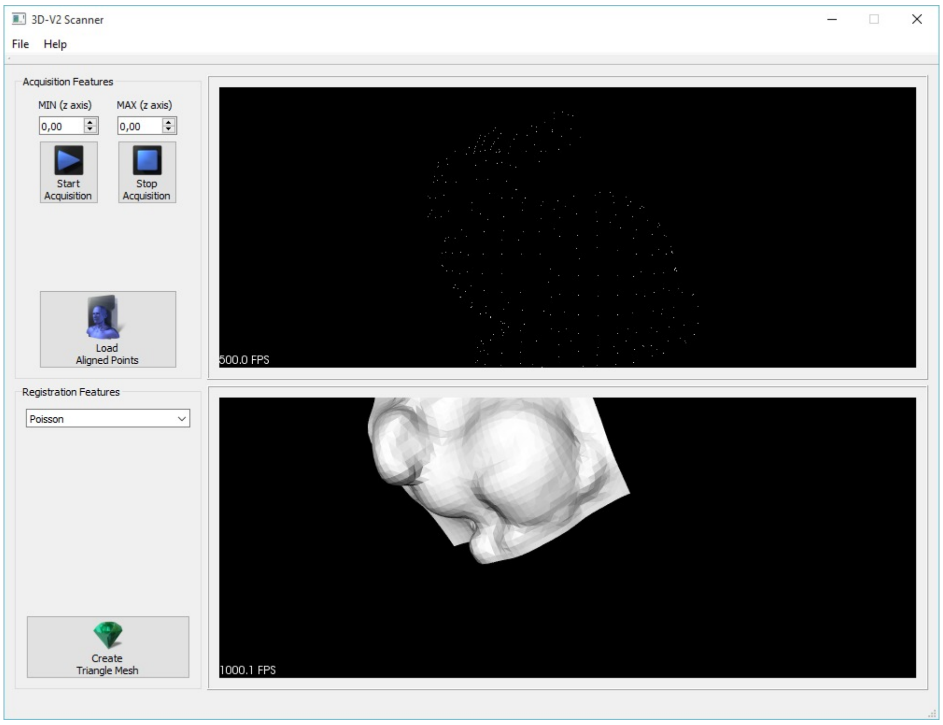

Figure 5.4: Poisson Triangulation

5.3.4 Comparison between two Projects

On 3D SCANNER project, the entire process of: acquisition, align and registration was, practically, developed into **ScannerWindow** (the QMainWindow class). Although, they created a **registration** class, that only provides the **loadData** function, to load all the point cloud (.pcd) files into their vector for align step. On the other hand, in our project, the point cloud processing was included into one single class, called **Point2Mesh** as explained on Section 5.3.

5.4 k2class class

The last class of this project was developed to manage the specificity from the sensor, in our case, Kinect V2. Since one of the proposes from this project is to turn the code well-class divided, the idea behind the creation of this class was to separate each intrinsic feature from each sensor (in case of in the future, more sensors were added).

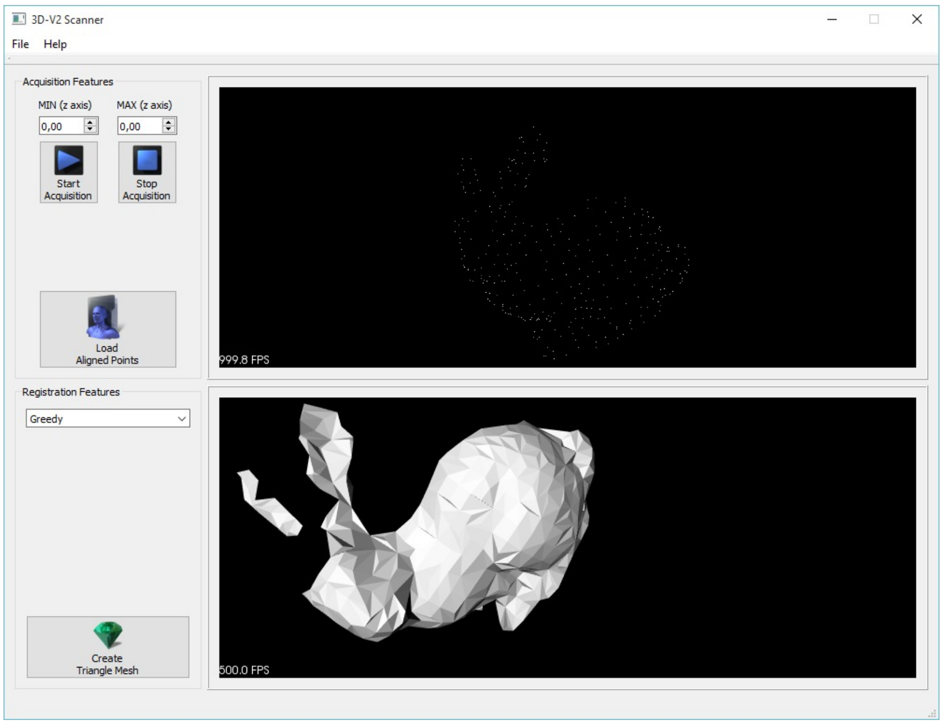

Figure 5.5: Greedy Triangulation

Since, for this project, no other device was used, the class is only composed by a function called **registration**, which connects to the previous explained function from **Point2Mesh** on Section 5.3, called **estimate_align**. As parameter of this function, it was delimited the set of point cloud collected by the Kinect V2 or loaded by the user.

Thus, a loop is used to read the point clouds in pair, which will be passed throw **estimate_align**, for alignment. The final result is composed by one single point cloud pointer and a file saved into the same directory as the project, called **"pointcloud.pcd"**.

Chapter 6

Results and Recommendations

6.1 Results

After a long development period, the product presented in this report provides 3D scanning and mesh processing approaches based on Kinect v2 from Microsoft. The software is divided into two main options: load/save 3D files and processing them and scanning objects using Kinect and processing it into a polygonal mesh.

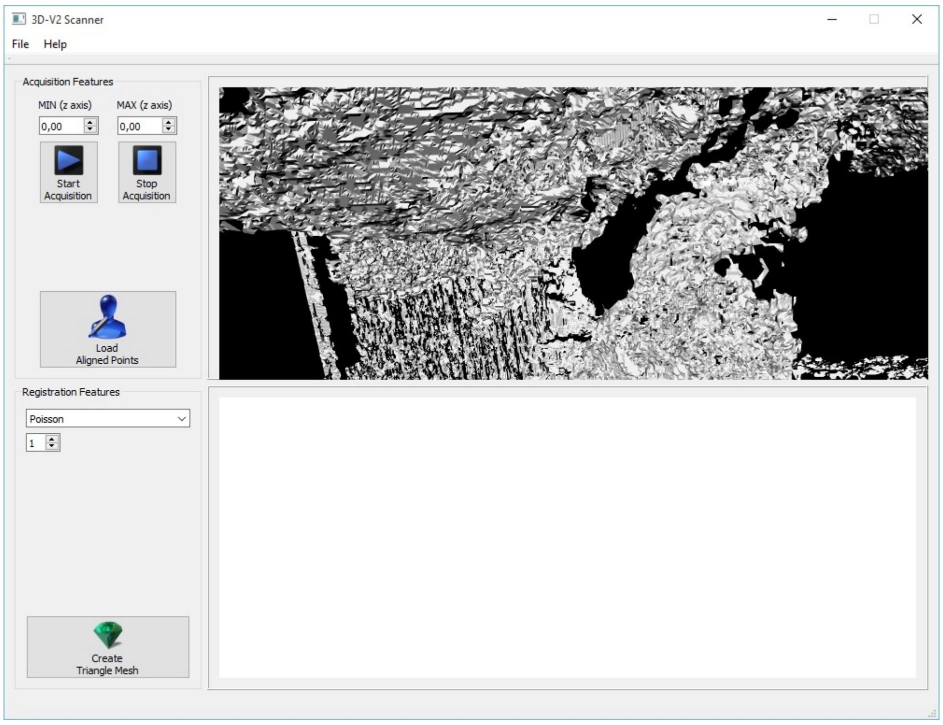

Figure 6.1: STL file loaded into the developed software

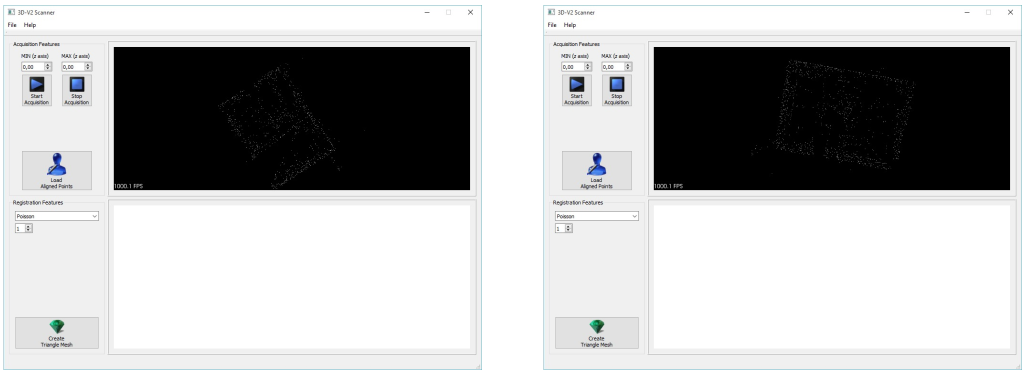

Figure 6.2: Left: first point cloud loaded; Right: second point cloud loaded

For this first part, the project allow us to load four different kinds of 3D files: **.vtk**, **.ply**, **.pcd** and **.stl**. In the case of **.pcd** files, meaning that the user loaded a raw point cloud into our software, we provide two options for polygonal mesh creation: Greedy and Poison, as detailed in Section 5.3.2. An example of a loaded **.stl** file is shown in Figure 6.1.

Another option developed into this project is the alignment of a set of loaded point clouds. Thus, the user can click on the menu option and select to load more than one **.pcd** file. After all the files were loaded, the user can click into the "Load Acquisition Point" button to execute the alignment function as shown in Figure 6.3.

After the alignment step, the last one is the triangulation processing, that can be achieved clicking on "Create Triangle Mesh" button and choosing the method on sidebar combobox. A triangulation result is shown in Figure 6.5

The second part of this product is composed by the scanning of point cloud by Kinetic V2, registration and triangulation steps. To create the scanned image, the user must click on "Start" button, and after the Viewer shows the point cloud, they can save each point cloud step clicking on the keyboard key "s". This allows the user to change the position of the object, without grabbing too many point cloud files. For each time the user press the button "s", one single file with **.pcd** extension will be saved into the same directory as the executable software.

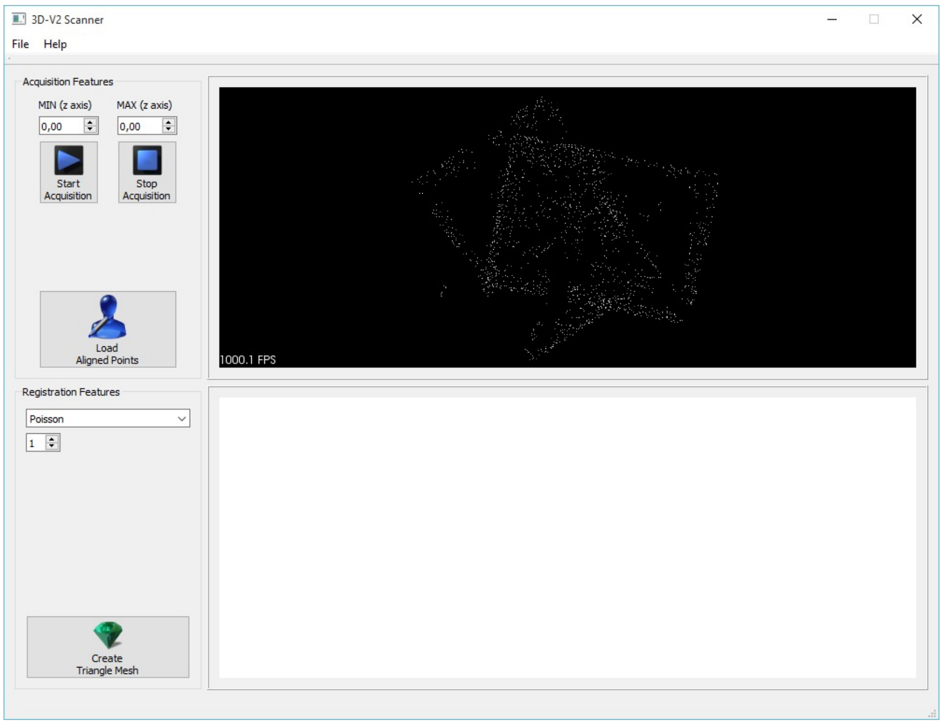

Figure 6.3: Alignment between two Point Clouds

6.1.1 Limitations

The limitation of this project is presented on the align and triangulation step from the grabbed point cloud. Even with two preprocessing mesh step, the software crashes when trying to compute the alignment step.

6.1.2 Recommendations and Future Works

To provide a best use of the scanning function, the best set up is to change the object position instead of Kinect v2 position and slowly grabb the point cloud by pressing the key.

For future works, the align function must be review since it only works with small point cloud size. Thus, it will be necessary to implement more functions for smoothing the raw point cloud grabbed by Kinect V2.

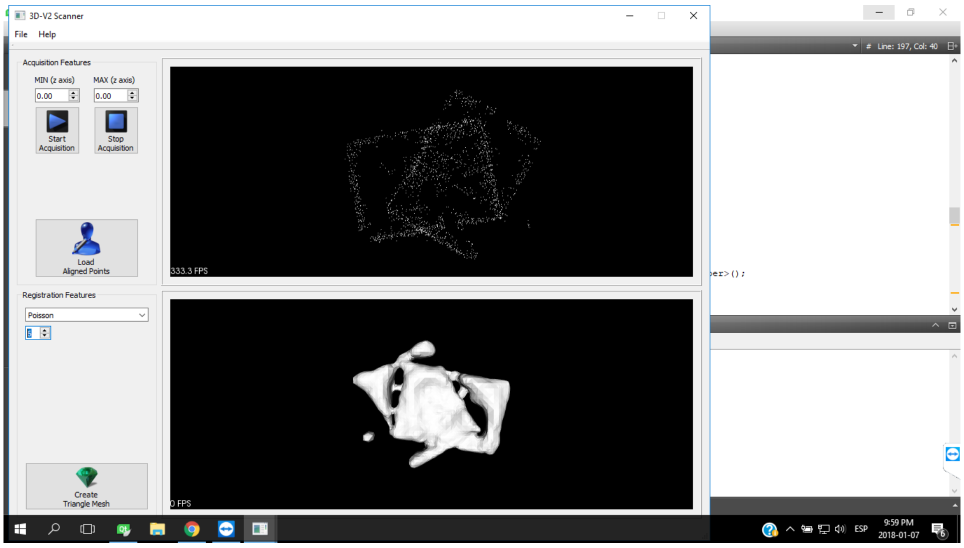

Figure 6.4: Triangulation of Aligned Point Cloud

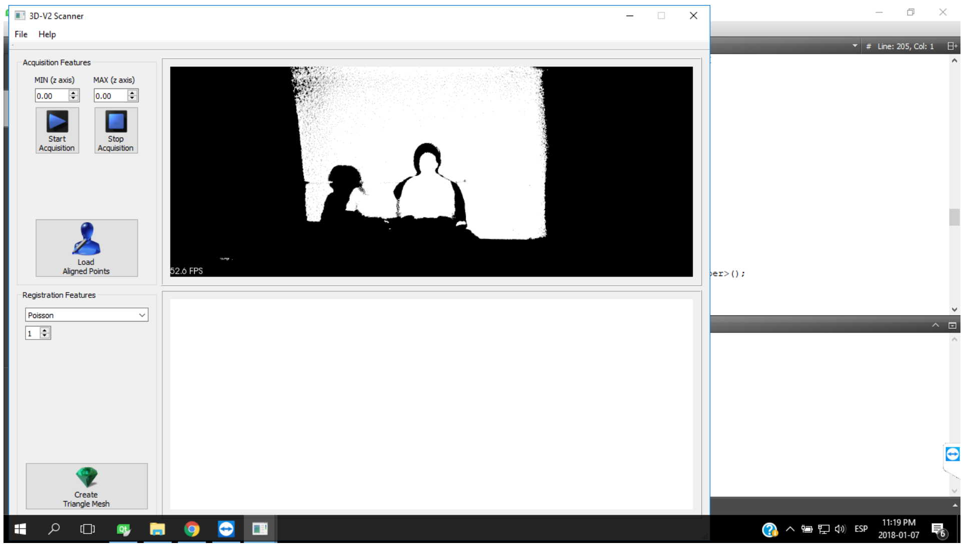

Figure 6.5: Grabbed Cloud Point from Kinect V2 Scanning

Bibliography

[1] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, *et al.*, "The digital michelangelo project: 3d scanning of large statues," ACM Press/Addison-Wesley Publishing Co.

[2] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), May 9-13 2011.

[3] D. X. a. Y. Song Tiangang, Lyu Zhou, "3d surface reconstruction based on kinect sensor," in *International Journal of Computer Theory and Engineering*, 2013.

[4] E. V.-F. J.-S. HGonzalez-Jorge, B Riveiro and P. Arias, "Metrological evaluation of microsoft kinect and asus xtion sensors," in *Measurement*.

[5] P. Flynn, "Formatting information: A beginners guide to latex (5th online ed.)," in *Cork: Silmaril*.

[6] Salma, "Uml class diagrams tutorial, step by step." `https://medium.com/@smagid_allThings/uml-class-diagrams-tutorial-step-by-step-520fd83b300b`, 2017.

[7] J. H. et al, "Surface reconstruction of point clouds captured with microsoft kinect," 2012.

[8] Qt Company, "Qt - cross-platform software development for embedded and desktop." `https://www.qt.io/`, 2017. Accessed 20 December 2017.