Project report:

MsCV - C++

By: Kevin Descharrieres - Romain Corsin

Professor: Jiang Cansen and Strubel David

January 7, 2018

Contents

I. GOAL OF THE PROJECT

A. Introduction

The last year students, by group of ten, made four different 3D scanner that we recovered to work on them. These different codes are accompanied by different reports with free access. We recovered everything from these files to try to understand by our own way how the scanners work. This project was manage here by two different students (Romain and Kevin) in less than ten weeks.

B. Constraints

For this project the goal was first to register data using a kinect, then to create the shape using the cloud of points we obtained before. There had two different constraints on this project, first the type of sensor was given so they had to find a solution to use only a kinect as 3D scanner which is cheap but with a really bad resolution. We had to find a solutions to find another way to register this point cloud to obtain a better shape with less noisy data. The second constraints was the time, the project had to compute the point cloud shape in less than 90 seconds.

C. Kinect

For this project we had to use the Kinect or the Intel R200, this is two sensors was used to register the point cloud.

FIG. 1: Kinect

The Microsoft kinect v2 is a time-of-ight sensor. It uses one infrared camera coupled with an infrared projector system. It has a 1080p (1920 x 1080) RGB camera and a 512 x 424 pixels depth

camera, both of which can stream at 30 frames per second. Its angular eld of view in horizontal and vertical planes is 60 and 70 degrees wide respectively and its sensing range is from 0.5 m to 4.5 m. The kinect v2 can not work well in sunlight because the infrared light from the sun.

FIG. 2: Intel R200

The Intel R200 is an infrared-projector assisted stereo camera. It uses a known pattern projected using an infrared laser projector to assist the stereo vision system to identify depth. It has a color camera with 1920x1080 pixels and a depth/IR camera with 640x480 pixels. The color camera can stream at 30 fps and the depth camera can stream at 30 and 60 fps. We hence ask the sensor for aligned capture at 30 fps. The angular eld of view of the sensor in horizontal and vertical planes is approximately 50 and 60 degrees respectively and its sensing range is from 50 cm to 2m. The sensor has problems perceiving dark textures, as well as depths of brightly lit scenes. The R200 is highly portable because of its small size, and uses a single usb for power as well as data. This is unlike the kinect which is bulky in comparison, and requires a separate power block.

D. Goal

The goal of the project is to modify the last year projects codes in view of improve the program of the scanners and take some parts of the codes to reconstruct our own project. In a first time, we has as order to debug the last year students programs to make them run into windows. The second great step of our work was to modify some parts of these codes to try to improve them a lot. We could change anything we wanted. From the interface to calling files or restart all the project, we had to choose what we will do. Another part of the work was to do some reverse engineering to understand the code we had available.

E. Our method

In our method, the first step was to read the different reports from last year to understand their way to perform the scanners. Then in a second step, we had to debug them and make them work on Windows before to start anything else in the project. That is what we tried to do in priority.

The third step was to find a way, no matter which, to improve these programs. After many problems, we decided to create a very simple interface easy to use. We wanted something very instinctive on the contrary of what was done on last year. The second thing that we decided to make is to improve the comments of at least one of the last year projects. This part was decided just after the step where we studied the project codes with a lot of troubles because there is a big lake of comments on them.

Let's see an example of the difference between before and now.

```
void TDK_Database::mf_StaticAddPointCloud(pcl::PointCloud<pcl::PointXYZRGB>::Ptr pointCloudPtr, QString pointCloudName)
{

    QString name;
    qDebug() << "Trying to add point cloud";
    mv_PointCloudsVector.push_back(pointCloudPtr);
    qDebug() << "Point cloud set";
    if("U1425_AUTOGENERATE" == pointCloudName){
        mv_NumberOfAutogeneratedPointClouds++;
        name = QString("TDK_CapturedPointCloud").append(QString::number(mv_NumberOfAutogeneratedPointClouds));

        mv_PointCloudsName.push_back(name);
        qDebug() << "Point cloud name set default " << QString("TDK_CapturedPointCloud").append(
                                                    QString::number(mv_NumberOfAutogeneratedPointClouds));
    }
    else{
        name = pointCloudName;
        mv_PointCloudsName.push_back( pointCloudName );
        qDebug() << "Point cloud name set " << pointCloudName;
    }
}
```

First you can read only pure code, the trouble here is to manage the quantity of information. The greatest part of variable are created previously so we don't even know their type looking only name and some part of the code or some function can be rude to understand by themself.

```
void TDK_Database::mf_StaticAddPointCloud(pcl::PointCloud<pcl::PointXYZRGB>::Ptr pointCloudPtr, QString pointCloudName)
{   //This function will add a point to the pointcloud


    QString name;                               //name for the new point
    qDebug() << "Trying to add point cloud";    //Sending a message to the debugger
    mv_PointCloudsVector.push_back(pointCloudPtr);  //We add the pointer to the vector
    qDebug() << "Point cloud set";              //new message
    if("U1425_AUTOGENERATE" == pointCloudName){ //if everything is fine
        mv_NumberOfAutogeneratedPointClouds++;  //we increase our static variable
        name = QString("TDK_CapturedPointCloud").append(QString::number(mv_NumberOfAutogeneratedPointClouds));
                                                //can create the name of the point using the static variable
        mv_PointCloudsName.push_back(name);     //we add the name to the data base
        qDebug() << "Point cloud name set default " << QString("TDK_CapturedPointCloud").append(
                                                QString::number(mv_NumberOfAutogeneratedPointClouds));
    }
    else{
        name = pointCloudName;  //we set the name of the point
        mv_PointCloudsName.push_back( pointCloudName ); //we suppress the name from the database
        qDebug() << "Point cloud name set " << pointCloudName; //we send a message
    }
}
```

After giving some explanation of the code, just giving the action performed by a line or a quick review for a function, the code is easier to understand, we can attach some work to the presentation of the code itself, work with spaces, name of variable or line skipping to let the person who is trying to read the code breath a bit.

II. DEBUGGING

In this section, we will introduce you how we tried to debug the last year students programs, the installation of the necessary elements to have to run them and the modifications to perform in the .pro files in each codes.

A. Study the reports

As said previously, the first step was to study the last year reports. by these reports, we tried to understand how the projects work. We saw, thanks to them, the project management, the time of each steps, the implementation of the code and so on. The problem with these reports was that it lacks many explanations about the code itself. So we tried to understand the codes by our own means using only the comments which are inside the codes.

This part was a little bit tedious because the reports has sometimes more than 80 pages. These pages was construct in a way to permit to someone which is already knowing how work each elements of the 3D representation. We spend many times to unscramble them and make sense for

us.

After this step, we decided to take as main example the code "3D KORN" to continue our project.

B. Installation of the necessary elements

We started debug every issues of the compiling of the "3D-Korn" program. After many hours of work with hundreds of problems, we understood that it was missing some elements to make it work or at least decrease the number of issues. The first step of this part was done, but we had many more issues. The necessary elements was not all installed but we did not know it at that time. After four weeks of an intensively work on the debugging, Roger gave to everyone a method to perform it with all missing elements and all paths to modify in the computer system.

The first library to install was the VTK one. "The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, image processing, and visualization. It consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python."[1] The VTK library permits to create vectors, some textures, some volumetric shapes and so on.

The second necessary installation concerned Visual Studio c++. Indeed, we had to install Visual Studio to visualize the 3D result. But to do it properly, we created the link between Qt and Visual. To do it, we had to install in parallel Qt 5.7 with MSVC 2015 for x64. When it was installed, the goal was just to find the compiler of Visual Studio in Qt. The second thing with this compiler was to run the program in release mode and not in debug mode.

To use the last year program and our one after, we had to install the software to use the kinect, KinectSDK. "The Kinect for Windows Software Development Kit (SDK) 2.0 enables developers to create applications that support gesture and voice recognition, using Kinect sensor technology on computers".[2]

For the calibration on the kinect, it was obliged to install intel RSSDK which is a depth camera

manager which permit the detection of elements in a picture, the reconstruction of 3D shape in an image, identify the presence of shapes or faces...

Another software to use is CMake. It is a cross-platform "production engine". It is similar to the Make program in that the software build process is entirely controlled by configuration files, called CMakeLists.txt in the case of CMake. But "CMake does not directly produce the final software, it handles the generation of standard build files: Makefiles on Unix, and Visual Studio project files on Windows."[3] This allows developers to use their preferred development environment as usual.

The last part to install and not the less important one is PCL 1.8.0 for Visual Studio 2015. The Point Cloud Library is a standalone, large scale, open project for 2D/3D image and point cloud processing. It contains the various processing for 3-dimensional point cloud that retrieved from sensors or 3-dimensional data files.[4]

C. Necessary paths

To build and run the program, it is necessary to change or add some path to the computer environment by adjusting the system environment variables (or similar paths):

- C:\Qt\Qt(version)\(version)\msvc2015 64\bin
- Add new variable QT_QPA_PLATFORM_PLUGIN_PATH
- C:\Qt\Qt(version)\(version)\msvc2015 64\plugins\platforms
- PCL_ROOT as C:\Program Files\PCL 1.8.0
- %PCL_ROOT%\bin
- %PCL_ROOT%\3rdParty\FLANN\bin
- %PCL_ROOT%\3rdParty\VTK\bin
- %OPENNI2 REDIST64%

These previous links will permit to the programs to find the right places of some files that it will need to run properly. Without these different links, we obtain several hundred of issues that we can not solve.

Then, in the .pro file, we had to modify some "include" paths to permit to the program to access at each needing files. For example we have :

- INCLUDEPATH += "C:\..."
- LIBS += "-LC:\Program Files\PCL 1.8.0\lib"

We have in the .pro file 186 links like the previous ones to tun the program. These 186 paths are the heart of the program because it is only thanks to them that the program, during the running operation, can find the needed files through all the files in the computer.

D. Success

After many attempts to build and run the project, we finally succeeded to do it as you can see on the following picture.

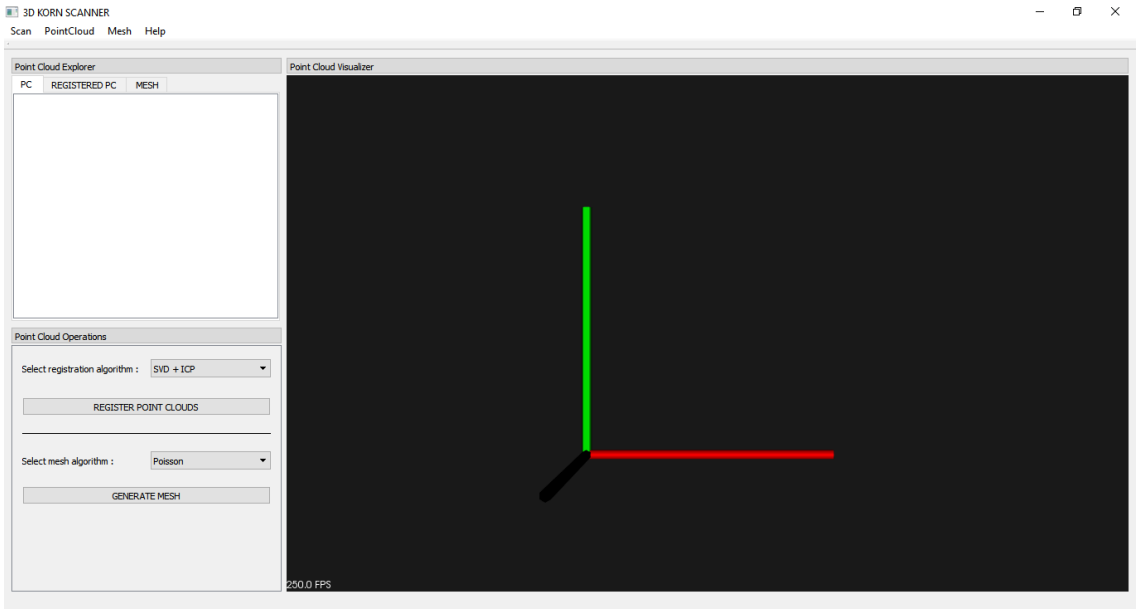

FIG. 3: Interface

On this interface, the manipulation is not instinctive. We don't know at the first sight how to manipulate it in view of put in the combine files to obtain a 3D result. The good way was to register the point clouds, in the second tab select them and in the third one to create the meshs. But we had some problems to display an image with the .ply and .pcd files. When we tried to display it, the program bugged and gave us an issue concerning a connection in the VTK file.

Few hours later, the only laptop which succeed to open the last year projects had a big bug, a black screen one and refused after that to restart. We tried to reinstall the system from a safeguard but it didn't work. We try also to demount the laptop to see is there were an unplug frame or anything else but everything was good. So we tried to reinstall all the necessary files on another laptop but without any success and the substitute computer to the dead one arrived only at the end of December. It was to late to try to restart all the project.

One of our goal was to improve the usage of its interface by creating several windows which will be open after each step. For example the first window will permit to open the .pcd files which will appear onto the second window to be selected and the third one will display the result to have a more instinctive interface. But after more reflections, we decided to create only one window with some explanations and which is easy to use.

E. Our work

We worked a lot on reverse engineering and spent a lot of time to understand the program of some other students. This amount of time is due to the lack of comments on their codes, we work on the 3D-Korn project which was the one from Umamaheswara, Savinien, Benjamin... The most of the code was well commented, but a big part of comment was lacking, like the database of the DTK. Because we couldn't make the program worked, we decided to work on the readability of the code. So we added to the code comments and some space to make the code readable. But because of some really complex structure, it become really complex to work on it. The reverse engineering is really complex due to the fact that it's not our own code.

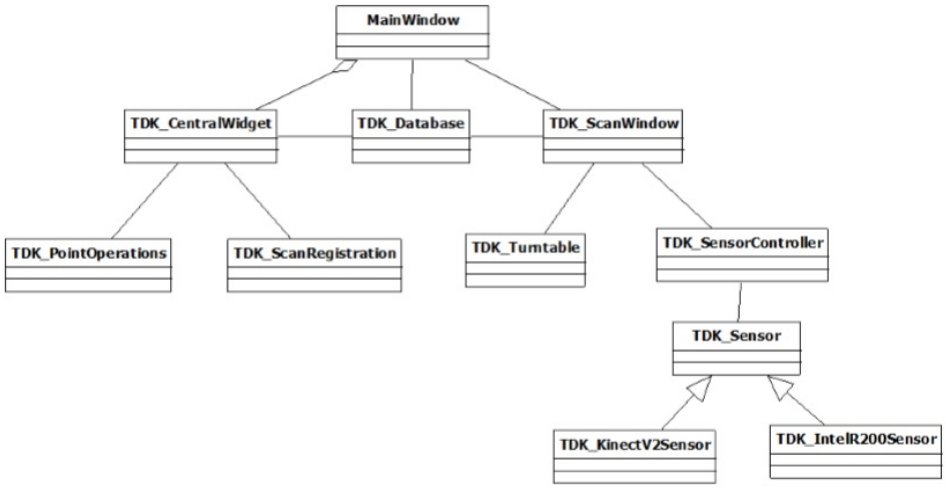

FIG. 4: Class diagram

Helped by the report they did, we can see the class diagram, which help a lot to understand the code itself, this code is divided between eleven classes, related each others.

- MainWindow, the principal class in every QT function using a graphic user interface, this class contain every slots, signals and widgets
- TDK CentralWidget will be the main widget.
- TDK Database, the database contain every data concerning the point cloud, names, positions of points...
- TDK ScanWindow, this is a derived widget which will contain every widget and create them
- TDK PointOperations will contain function which will proceed on points like find links between points.
- TDK ScanRegistration will be used to register pointcloud.
- TDK Turntable will manage the turn table we use to make the person turn and register the shape.
- TDK SensorController used to detect and make run sensors.

- TDK Sensor which is a virtual class to create the last two classes
- TDK KinectV2Sensor derived from sensor, managing the kinect.
- TDK IntelR200Sensor derived from sensor to manage the Intel R200 sensor.

III. GRAPHICAL USER INTERFACE

As said previously, the interface will be composed of only one window. We will describe it into three different steps. The first one will be how to call the different files, the second one how to choose the good extension and the last one how to display the final result.

A. Call files

To call the different files, we put a little button on the interface. This one permits only to open a dialog box to choose the different files. Thanks to the GUI, the creation on the interface is easier than in line of code as we found in the previous projects. Lets take the example with the button just after.

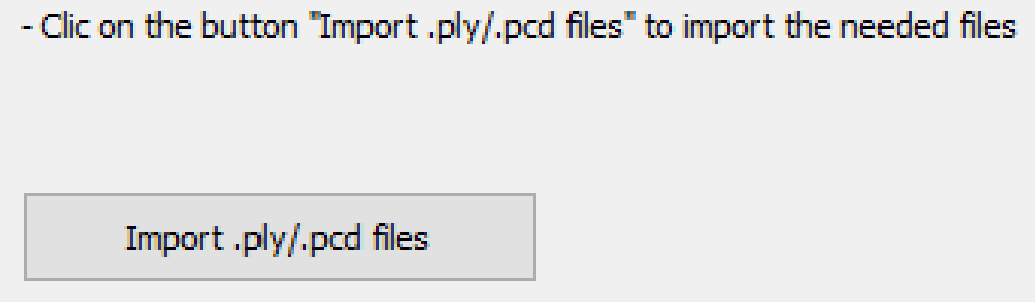

FIG. 5: Button

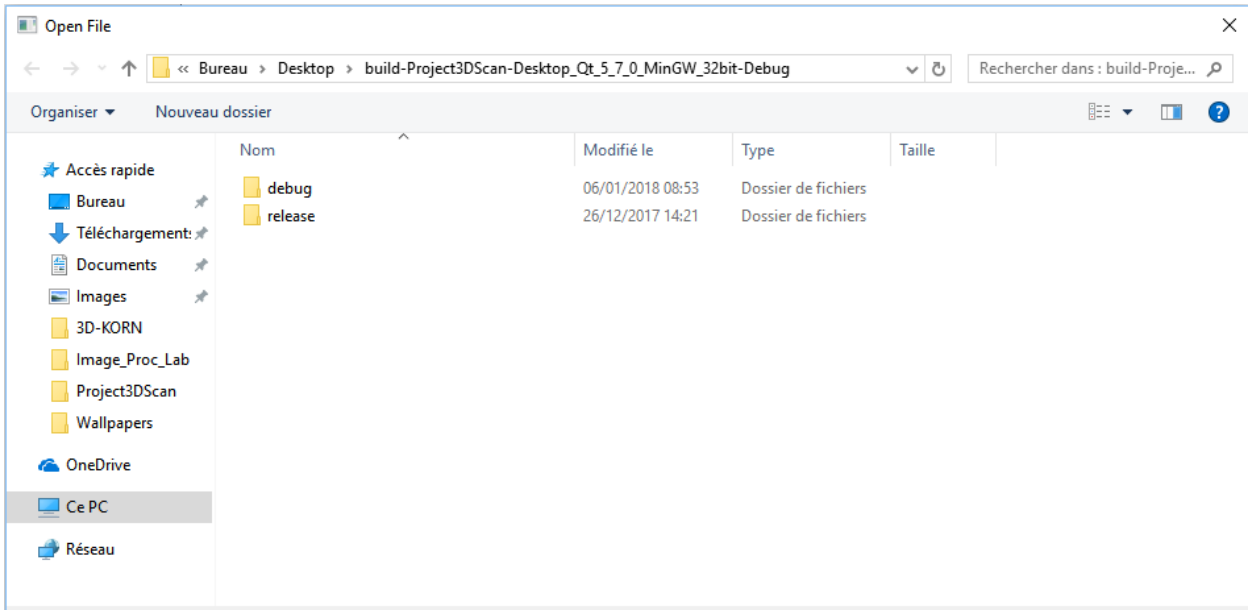

FIG. 6: DialogBox

Thanks to this DialogBox, the user can open every files with a specific type of extension as we can see in the following part.

B. Select necessary files

To use the scanner, the goal is to choose a specific type of files to use only one of them. To do that, we decided to give the choice to the user using some radio buttons.

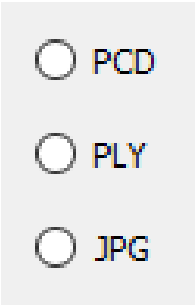

FIG. 7: Radio Buttons

Here the .jpg is present only for the example because the .pcd and .ply do not work on our project. So, the first step for the user is to choose the format to use by click on one of the radio

buttons. By this, when he will click on the button to open the dialog box, the default extension will correspond to his choice.

For example, if the user choose the .ply format, the dialog box will automatically search the same extension files and the others will not appear.

The part of code to realise this operation is the following one with the example of ply format :

```
if(ui->radioButton->isChecked() )
{
    QString fileName = QFileDialog::getOpenFileName(this,
                                                    tr("Open File"),
                                                    QDir::currentPath(),
                                                    tr("Images (*.ply)"));
    //And display the image
}
```

This "if" condition is inserted into the slot of the previous button. Indeed, when the user is clicking on the button, the program directly checks if the radio button are checked or not. If it find a checked one, it takes the corresponding format as reference for the following steps.

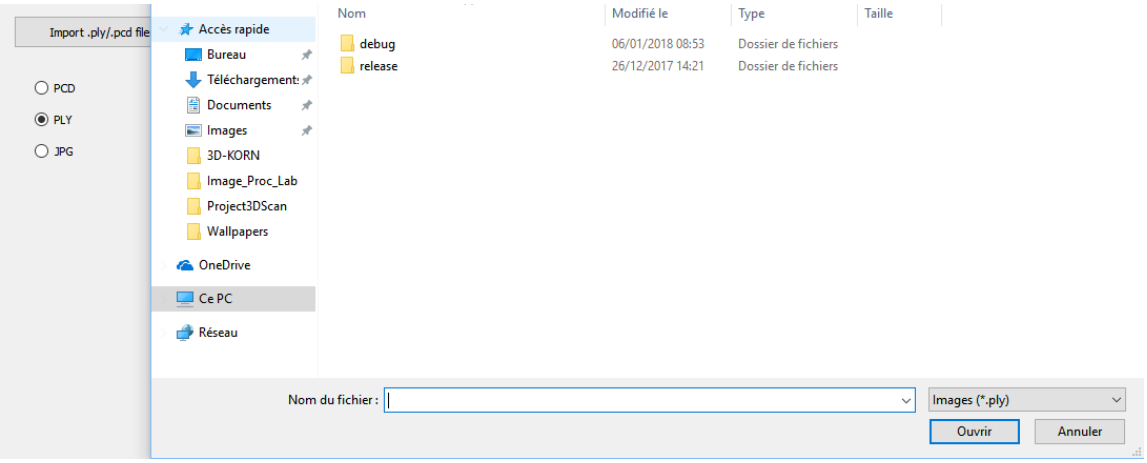

FIG. 8: Choice of the user

C. Final result

For this last part, we will take as functioning example a .jpg format picture. Lets start with the beginning, as previously, the user will open the interface and choose the .jpg format.

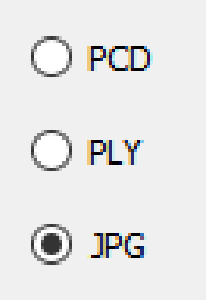

FIG. 9: JPG format

After clicking on the button, the dialog box is opening and the user choose the wanted image. He is clicking on OK and the picture will be display directly in the graphics view part.

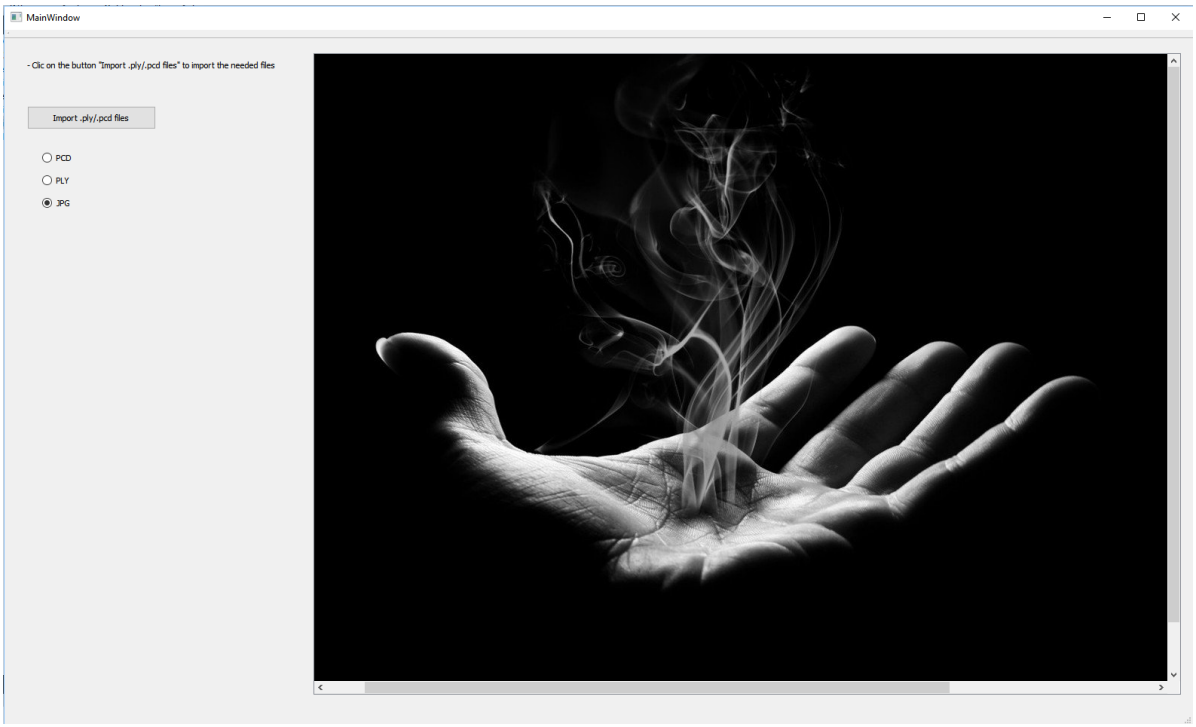

FIG. 10: Final result

This manipulation is possible thanks to the following part of code :

```
scene = new QGraphicsScene(this);//creation of the scene
ui->graphicsView->setScene(scene);//set the scene on the graphicsView
```

```
QImage image(fileName);
        QPixmap pixmap(fileName);
        scene->addPixmap(pixmap);
        ui->graphicsView->setScene(scene);
        ui->graphicsView->show();
```

The first part of the previous code permits to create the scene which will receive the correct files. The second part permits just to display the files (here a jpeg picture) into the QGraphicsView.

IV. CONCLUSION

As a conclusion, the project was based on the last year students projects. We had to try to improve them in view of create a software easier to manipulate and faster to compile. The first step was to run the last year projects under Windows. This step was complicate because we didn't have the good informations at the beginning but with lots of work and after few weeks we succeeded in this task. At that time, we had a big problem in the realization of the project. The only laptop that we had which ran the projects died without any chance of repair.

Without waiting, we tried to run the program on other laptops but it never worked. So we decided to improve the programs by another way. The first step was the creation of another interface which could easily replace the last year ones. This interface is constructed on a very simple manner. It should be easy to graft it to the first projects.

The second step was to improve the comments of the last year projects due to the fact that their is a big lake of them. After understanding their code (mainly the 3D KORN one), it was a little bit easier to do them. We added some comments as much as possible. Now, the 3D KORN code is more readable by an uninitiated person to the 3D scanner project. As this we could see how it could be difficult to understand a code without any explanation on this last. The amount of time

we spent to understand this could be divided by two or more with some proper explanations.

V. TO GO FURTHER

To go further, we can imagine this type of scanner with a better quality camera in view of after the 3D representation smooth the results. Once the result is smoothing, we could use a 3D printer to create a little object with the image of someone. To do it, we could add a command and create the link between the scanner and the printer in only one step.

Another way to improve this project should be to do it not only for one object or person but for a complete environment. For example, instead of taking only a guy which is standing and waiting, we could represent him inside a room or in a garden.

We could create a track to move the kinect around the person to be closer, using an algorithm to manage the travel of the track.

Now, can we improve it to create a 3D representation in live?

---

[1] https://www.vtk.org/

[2] https://www.microsoft.com/en-us/download/details.aspx?id=44561

[3] https://fr.wikipedia.org/

[4] http://unanancyowen.com/en/pcl18/