3D Scanner

Project Defense

PROJECT SUPERVISOR:

PROF. YOHAN FOUGEROLLE

CANSEN JIANG

Team members:

- Yu LIU
- Chunxia LI
- Anirudh PULIGANDLA
- Darja STOEVA
- Gopikrishna ERABATI
- Hassan ZAAL
- Avinash NARAYANA
- Shubham WAGH
- Ziyang HONG
- AbdelRahman ABUBAKR

Outline

1. Problem statement
2. Introduction
3. Project management
4. Technical parts
5. Qt classes
6. Turning table
7. Conclusion

Problem Statement

- 3D Scanners are quite proficient today and have become familiar in our everyday lives.
- They are not restricted to only academic and industrial applications.
- **Aim :** To make a home-made acquisition and processing software generating accurate 3D models using off-the-shelf Kinect v2 camera.

Introduction



- Implementation of fully-functional 3D scanning software outputting watertight triangulation meshes
- Targets home-made and quick scanning (acquisition time less than 90 seconds)
- Stationary Kinect V2 for depth data acquisition
- Available Point Cloud Library used as the main resource for depth data processing

Project Management

Internal Organization :

1. **Scanning Team** – interfacing Kinect v2, grabbing PCD frames, registration.

2. **Meshing Team** – normal estimation, upsampling, meshing techniques, mesh smoothing.

3. **GUI Team** – QVTK widgets, user friendly interface designing.

4. **Hardware Team** – rotating turntable construction, serial communication .

Project Management

Weekly Meetings & Reports:

1. Several meetings and conversations with the tutors (October – December).

2. Weekly meetings with the members of the group.

3. Weekly report submissions.

4. Feedbacks on weekly reports by the tutors.

Project Management

Initial weeks, **Meshing** team concentrated 30% of the original work on meshing methods and 70% of the work to support **Scanning** team.

Later weeks, **Scanning** team concentrated 30% of the original work on registration methods (improvising results) and 70% of the work to support **Scanning** team.

Scanning



Kinect v2 :

1. Time-of-flight sensor technology.

2. Higher depth fidelity.

3. Improved 3D visualization.

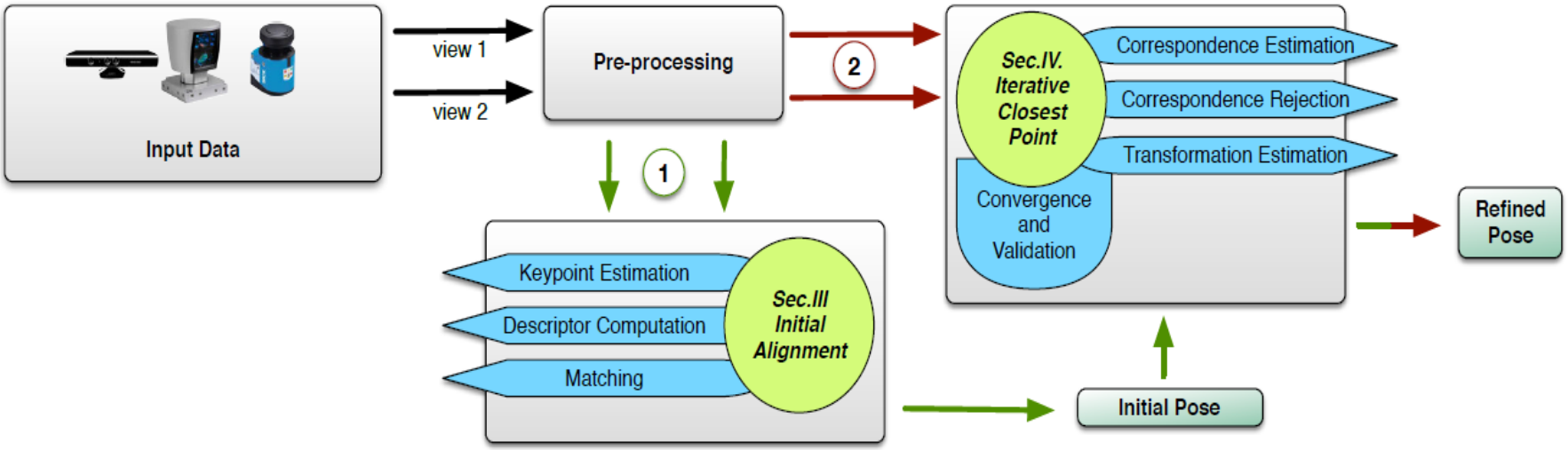
4. Improved ability to see smaller objects and all objects more clearly.

Scanning

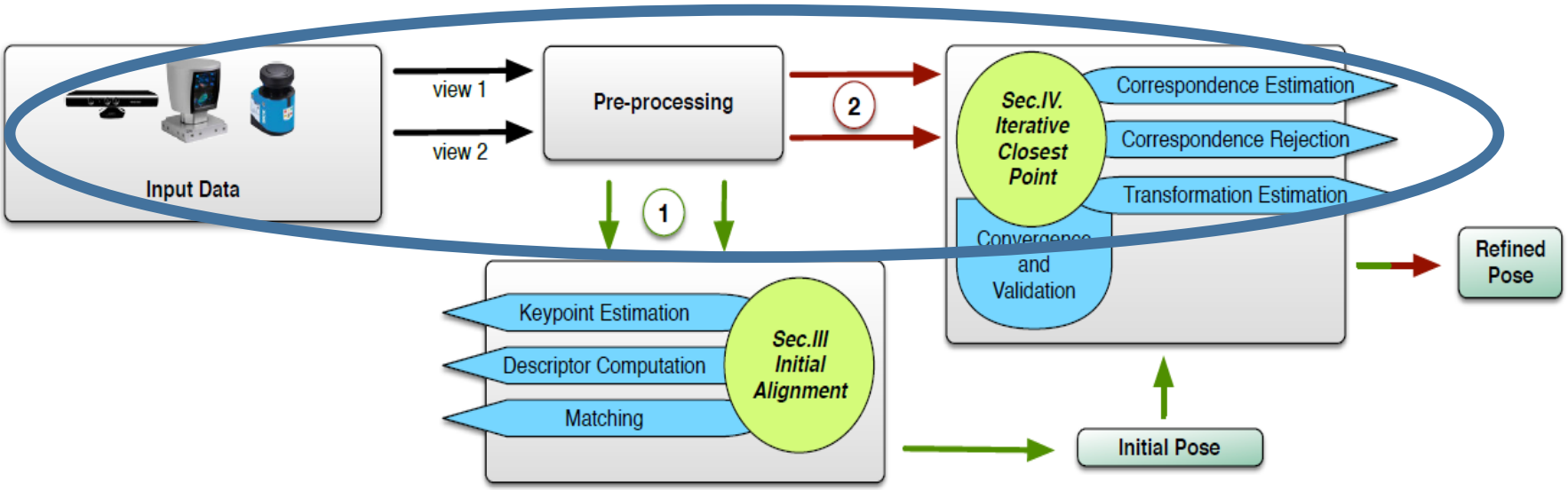
Kinect v2 Interfacing :

- PCL doesn’t have grabber for input data from Kinect v2. (OpenNI does not provide any support).
- Kinect2Grabber - Grabber to input data from Kinect v2 based on only Kinect SDK v2.
- Callback function is defined.
- Returns shared pointer of point cloud to outside.
- Real-time display of point cloud is retrieved.

Registration Pipeline



Registration Pipeline

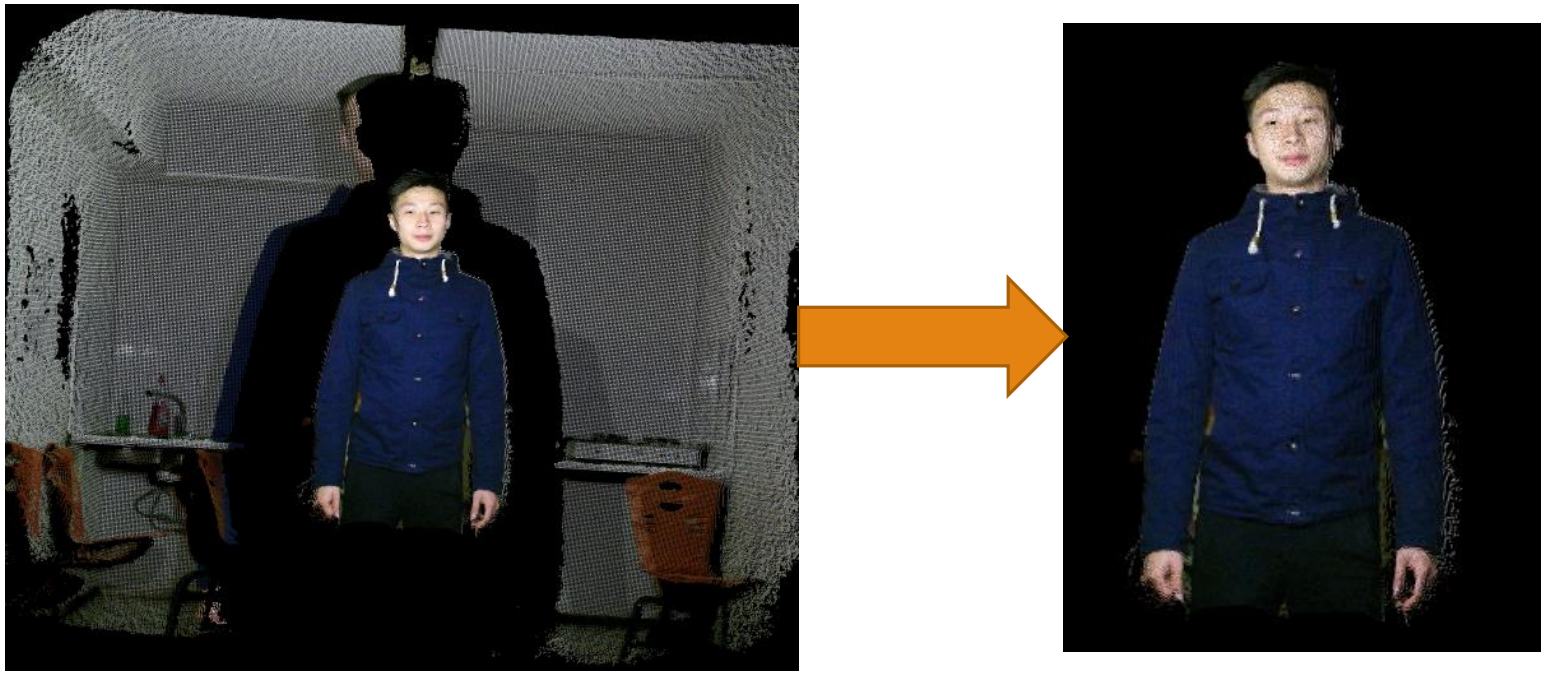


Preprocessing – Pass-Through Filter

- Segmentation using pass-through filter (X, Y, and Z directions)
- Significantly reduces number of points to be processed

```
pcl::PassThrough<PointT> pass;
pass.setInputCloud (CloudIn);
pass.setFilterFieldName ("z");
pass.setFilterLimits (ZLow , ZHigh);
pass.filter (*CloudFilteredZ);
```

Preprocessing – Pass-Through Filter

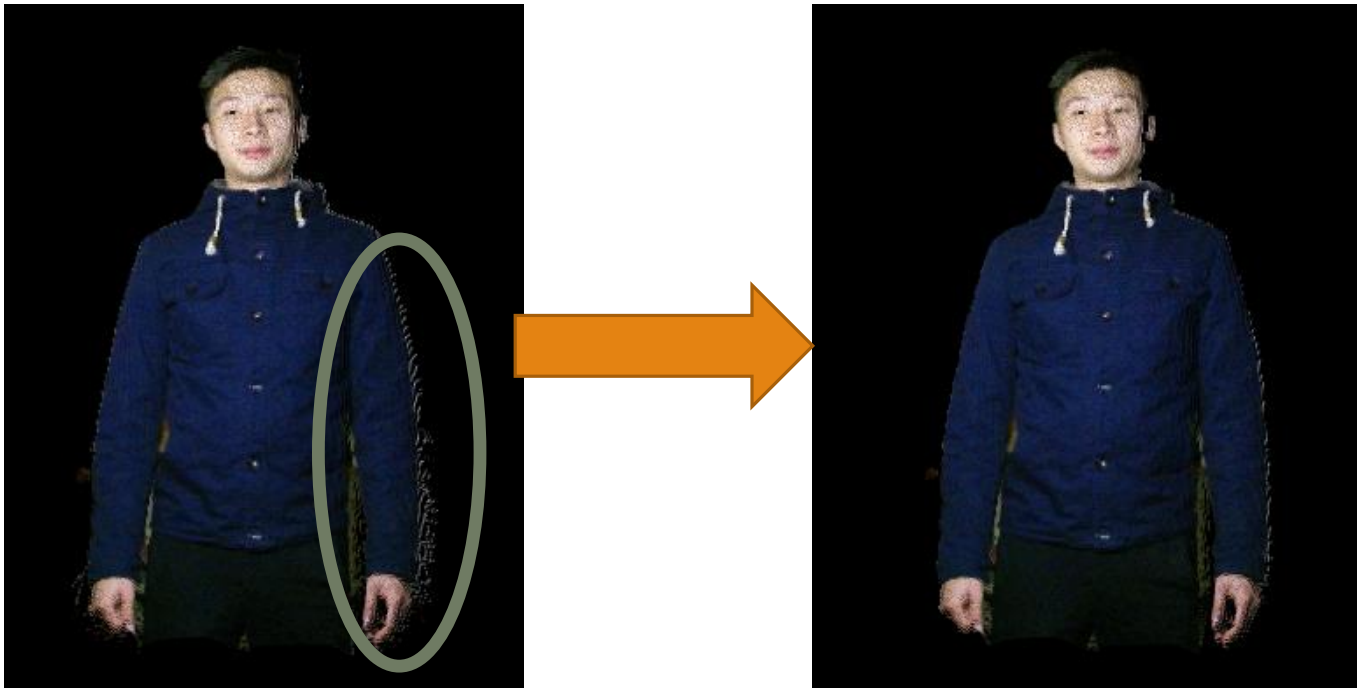


Preprocessing – Outlier Removal

- Target sparse points significantly far away from the main cluster
- Statistical outlier filter (analysis on each point's neighbor)

```
pcl::StatisticalOutlierRemoval<PointT> sor;
sor.setInputCloud (CloudIn);
sor.setMeanK (NumNeighbor);
sor.setStddevMulThresh (Threshold);
sor.filter (* CloudFilteredZ);
```

Preprocessing – Outlier Removal



Downsample

- Further reduction of number of points
- Voxel grid filter (approximation using centroid in a grid)
- Reduce processing time for ICP

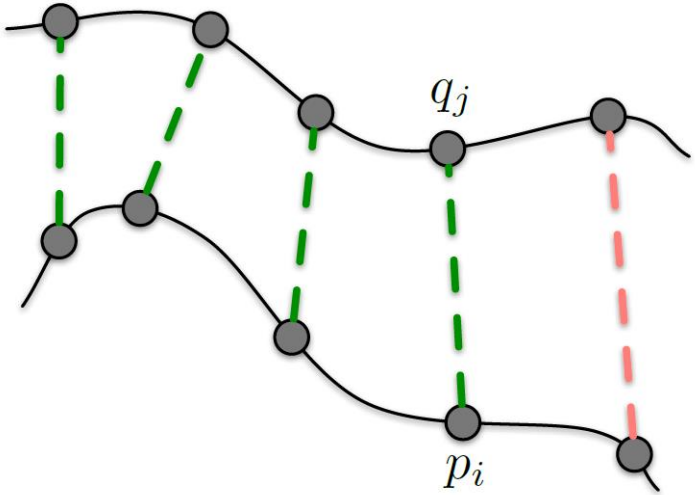
```
pcl::VoxelGrid<PointT> vox;
vox.setInputCloud (CloudIn);
vox.setLeafSize (LeafSize, LeafSize, LeafSize);
vox.filter (*CloudDownsampled);
```

Alignment – Iterative Closest Point (ICP)

- Appropriate approach for data with rich overlaps
- Standard ICP V.S. ICPwithNormals
- One ICP V.S. Multiple ICPs

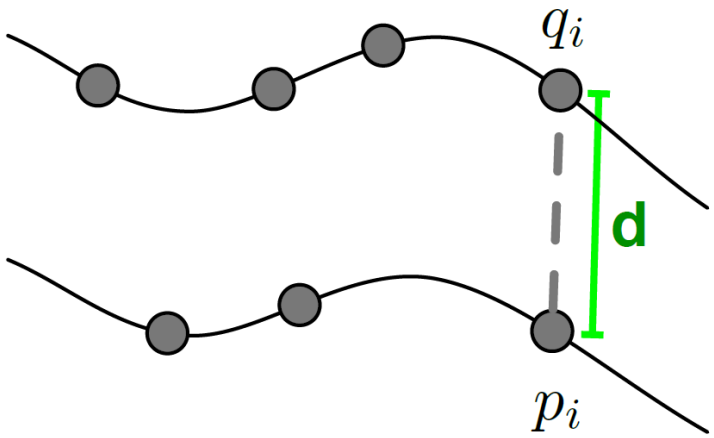
Correspondence Estimation

- Searches for closest points between source and target points
- False correspondence rejection based on distance criteria



Standard ICP: Error Metric

•Point-to-point Euclidean Distance



$$E_{\text{point-to-point}}(\boldsymbol{T}) = \sum_{k=1}^{N} w_k ||\boldsymbol{T}\ \boldsymbol{p}_k - \boldsymbol{q}_k||^2$$

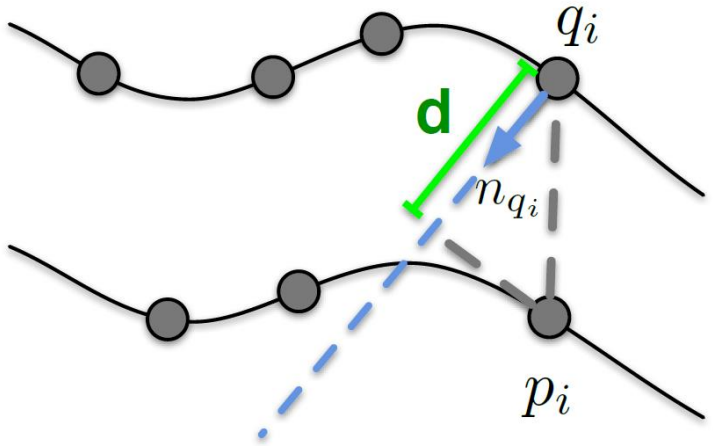
Standard ICP Code Snippet

```
   PointCloud::Ptr ICP(PointCloud::Ptr src, PointCloud::Ptr tgt)
 {

        pcl::IterativeClosestPoint<PointT, PointT> icp;
        icp.setMaxCorrespondenceDistance (MaxDistance);
        icp.setTransformationEpsilon (1e-8);
        icp.setMaximumIterations (Iterations);
        icp.setInputSource(src);
        icp.setInputTarget(tgt);
        icp.align(*transformedSrc);
}
```

ICPwithNormals: Error Metric

•Point-to-plane Euclidean Distance



$$E_{\text{point-to-plane}}(\boldsymbol{T}) = \sum_{k=1}^{N} w_k \left((\boldsymbol{T}\,\boldsymbol{p}_k - \boldsymbol{q}_k) \cdot \boldsymbol{n}_{\boldsymbol{q}_k} \right)^2$$

ICP with Normals Code Snippet

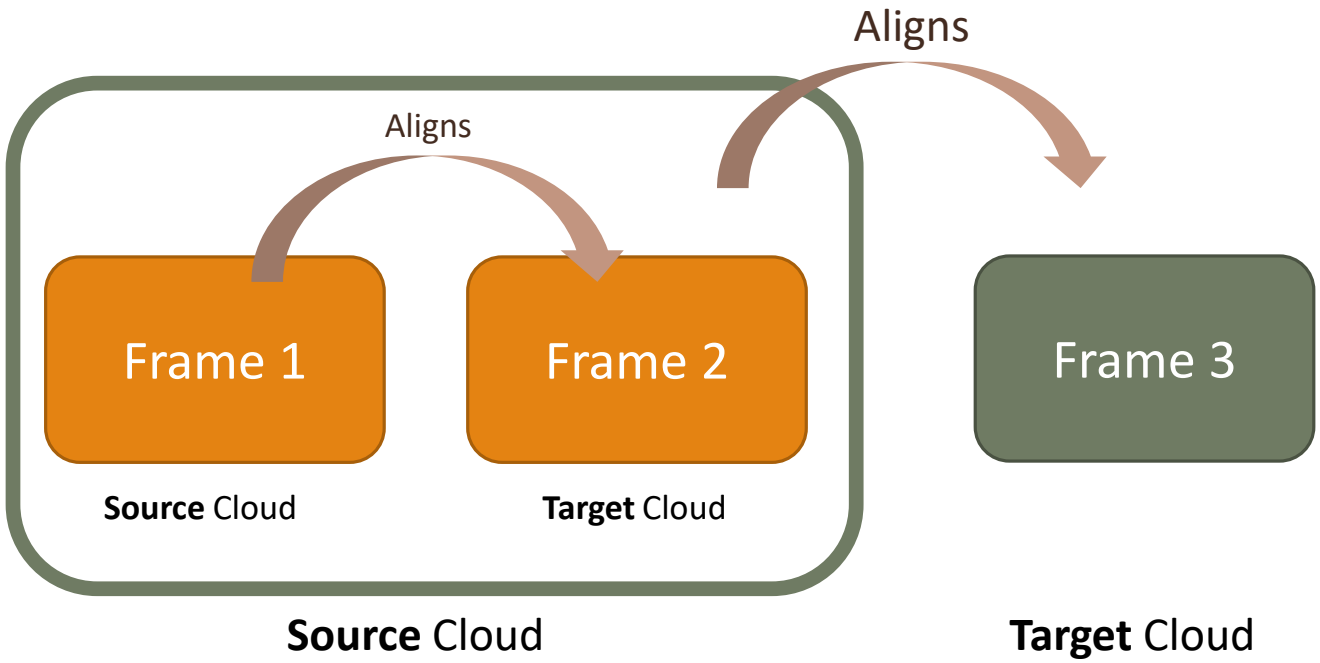
```
PointCloud::Ptr ICPNormal(PointCloud::Ptr src, PointCloud::Ptr tgt) {

        pcl::IterativeClosestPointWithNormals<PointNormalT, PointNormalT> icpn;
        icpn.setMaxCorrespondenceDistance (MaxDistance);
        icpn.setTransformationEpsilon (1e-8);
        icpn.setMaximumIterations (Iterations);
        icpn.setInputSource(src_with_normals);
        icpn.setInputTarget(tgt_with_normals);
        icpn.align(*transformed_src_with_normals);

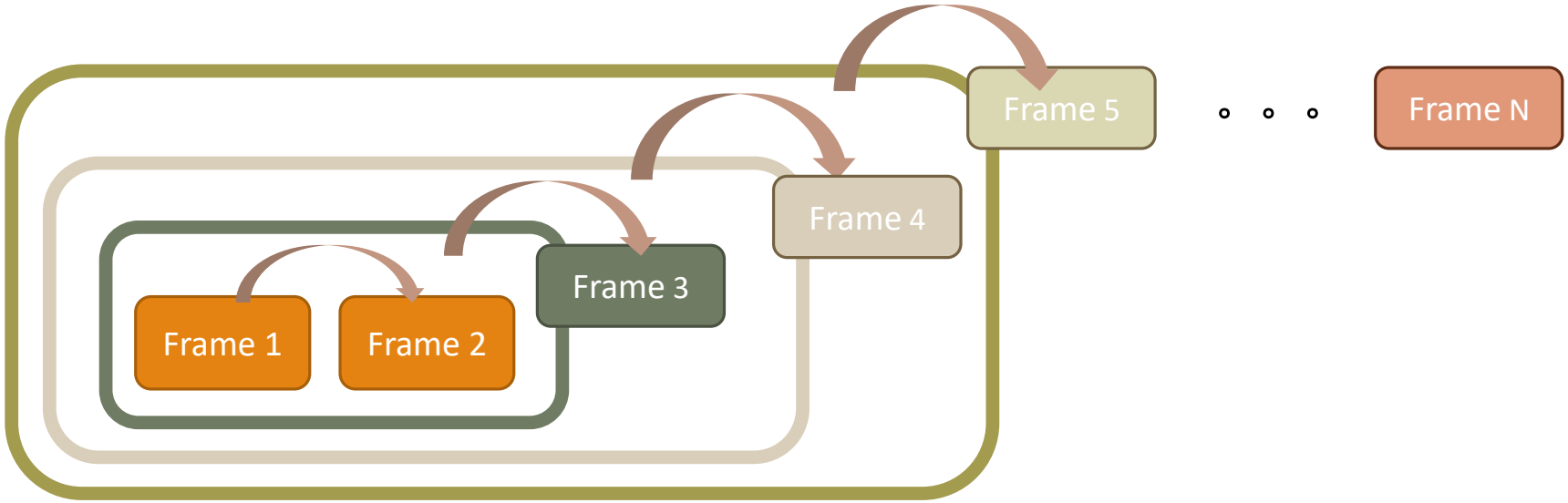
        Eigen::Matrix4f transform = icpn.getFinalTransformation ();
        pcl::transformPointCloud (*src, * transformed_src, transform);

}
```

Incremental ICP-based Registration



Incremental ICP-based Registration



Light downsampling in between for speed enhancement

Alternative: Two ICPs in Sequence

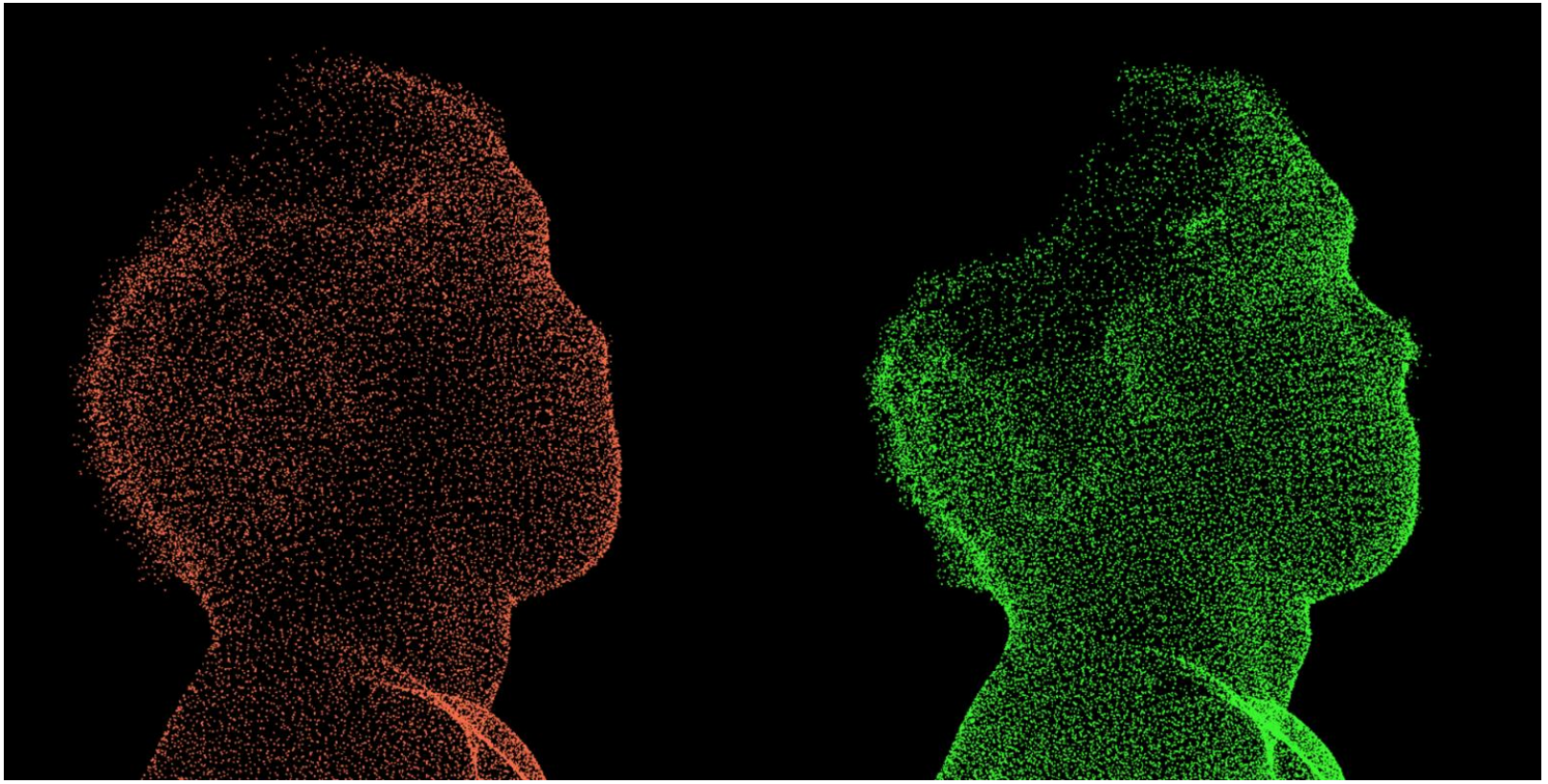
- First ICP for initial alignment in case of limited overlapping (looser false correspondence rejection criteria)
- Second ICP for refining alignment (strict rejection criteria)

Smoothing

- Moving least squares surface reconstruction smooths noisy data
- Multi-layer to single surface
- Crucial step for proper meshing



Smoothing – Noisy Data V.S. Details



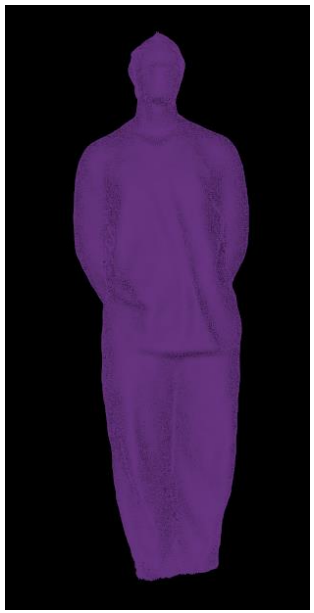
Greedy Projection Triangulation

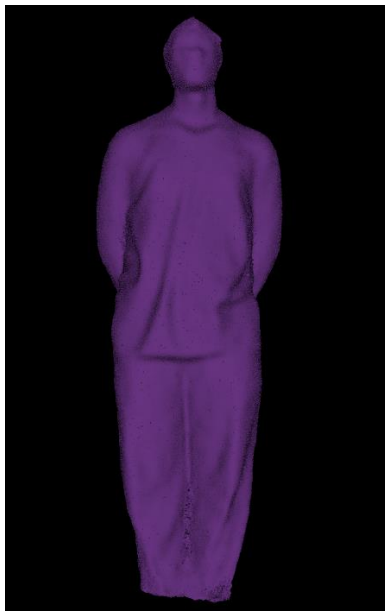
- For Reliability and accuracy of the mesh three topics are important:
- Noise
- Holes
- Details

Meshing

- A mesh can be grown and extended by maintaining the list of points until all possible points are connected
- Works best until the surface is locally smooth
- Computes all distances, sorts them and examines each pair of points in length and compatibility, then edges are created.

Greedy Projection Triangulation



Code snipet

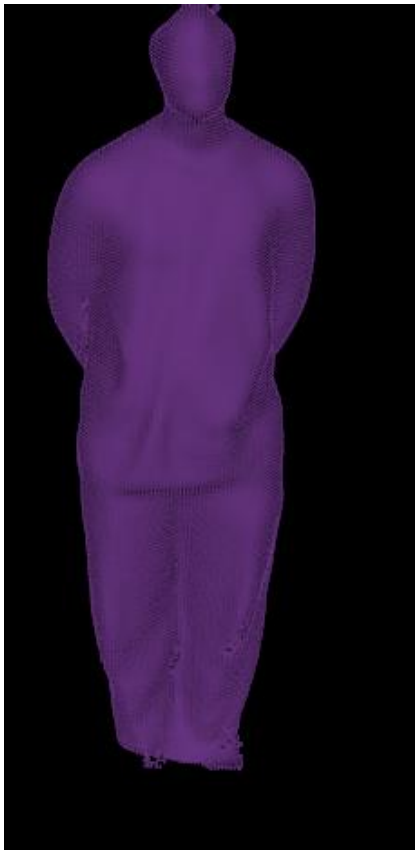
```
pcl::GreedyProjectionTriangulation<pcl::PointNormal> gp3;

gp3.setMaximumNearestNeighbors (n);
gp3.setMaximumSurfaceAngle(S max);
gp3.setMinimumAngle(Min angle);
gp3.setMaximumAngle(Max angle);
gp3.setNormalConsistency(Boolean);
```

Grid Projection

- The algorithm tries to create surfaces that lacks boundaries and proper orientation
- Works best for largest meshes(Flat) and with out too much noise.
- It creates continues surface unlike greedy projection triangulation.

Grid Projection



SNIPET:

```
pcl::GridProjection<pcl::PointNormal>
gp;
gp.setSearchMethod(kdtree Ptr);
gp.setResolution(res);

gp.setPaddingSize(3);
```

Poisson Surface Reconstruction

Concept:

Combines local and global fitting methods.

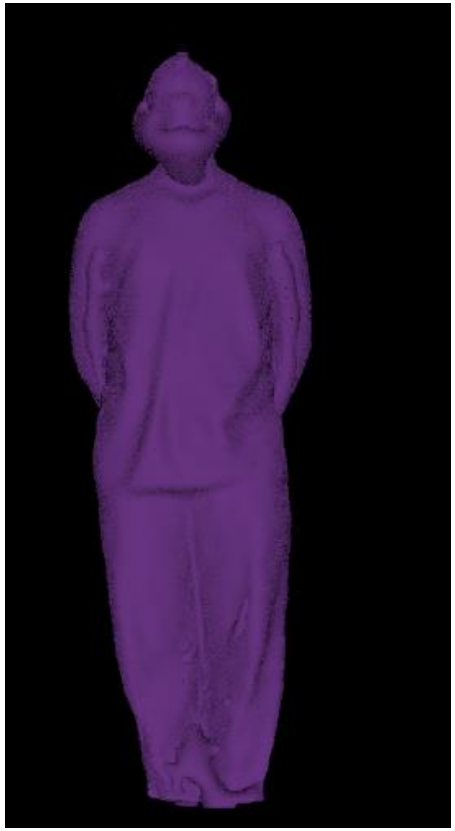
Works best with noisy data points and can recover fine details

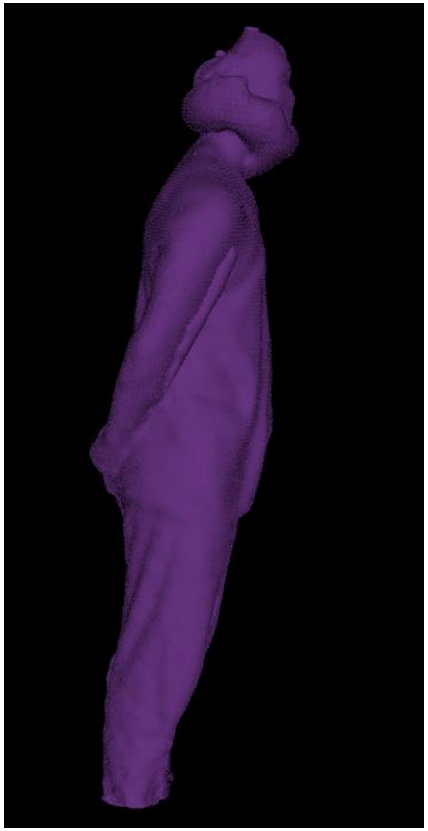
Reconstructs a water tight model by using Indicator function and extracting the isosurface.

Limitation:

Depends on the acquisition of the data points

Poisson Reconstruction

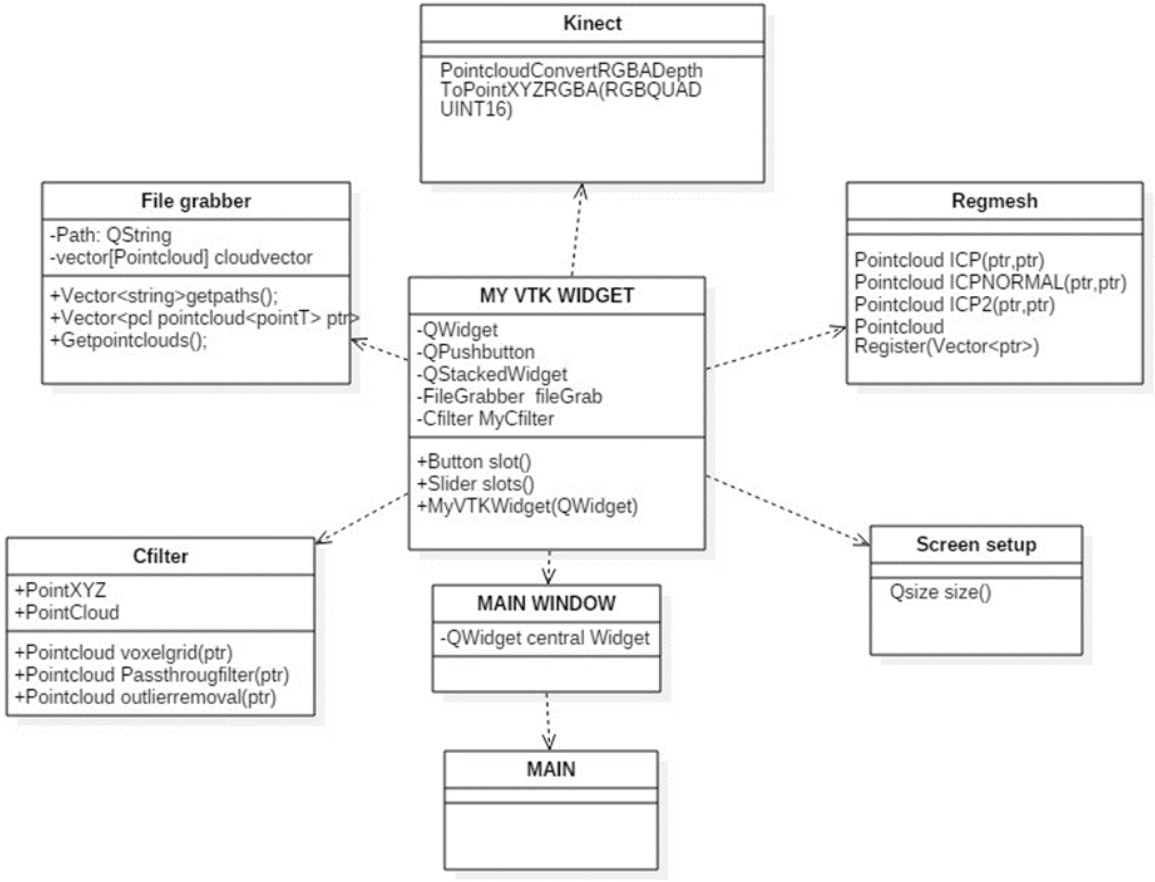




Poisson code snipet

```
pcl::Poisson<pcl::PointNormal>poi;
poi.setDepth(12);
poi.setScale(1.2);
poi.setSolverDivide(8);
poi.setIsoDivide(8);
poi.setSamplesPerNode(2);
```

Class diagram

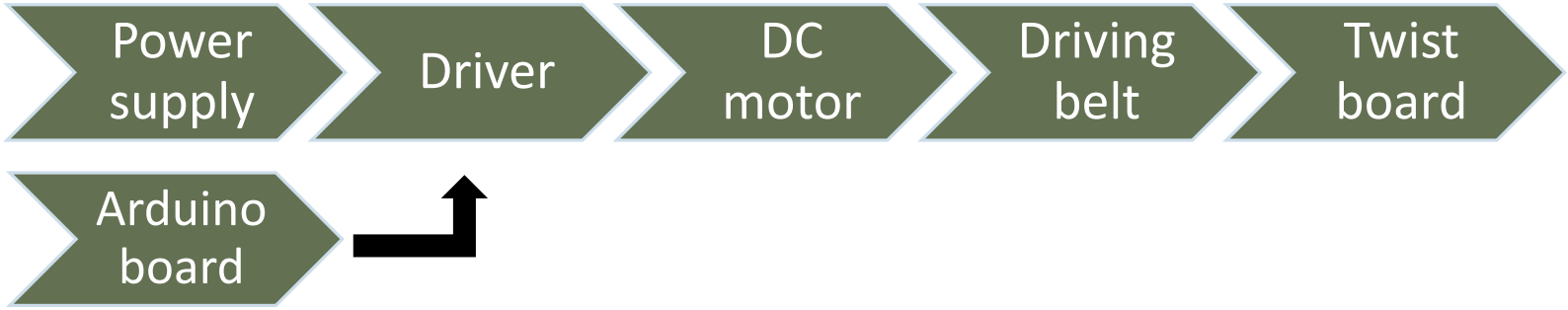


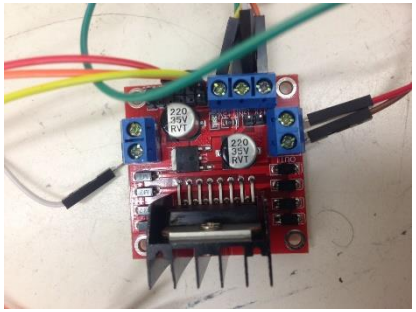
Qt CLASSES

- Qwidget
- QStackWidget
- QListWidget
- QGridLayout
- Radiobutton
- QVTKWidget

```
#include <QWidget>
#include <QVTKWidget.h>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/io/io.h>
#include <pcl/visualization/cloud_viewer.h>
#include <vtkSmartPointer.h>
#include <vtkRenderWindow.h>
#include <QDebug>
#include <QWidget>
#include <QListWidget>
#include <QListWidgetItem>
#include <iostream>
#include <QVBoxLayout>
#include <QPushButton>
#include <QSize>
#include "screensetup.h"
#include <QDebug>
#include <QProgressBar>
#include "filegrabber.h"
#include <vector>
#include <QString>
#include <QDebug>
#include "kinect2_grabber.h"
#include "cfilter.h"
#include "cinout.h"
#include <QStackedWidget>
#include <QSlider>
#include <QLabel>
#include <QGridLayout>
```

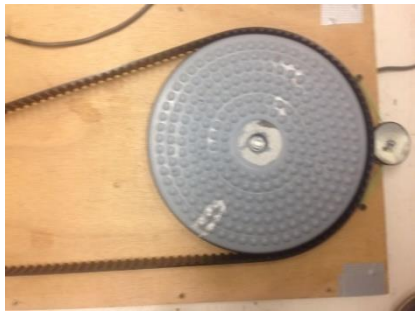
Turning Table (Hardware)





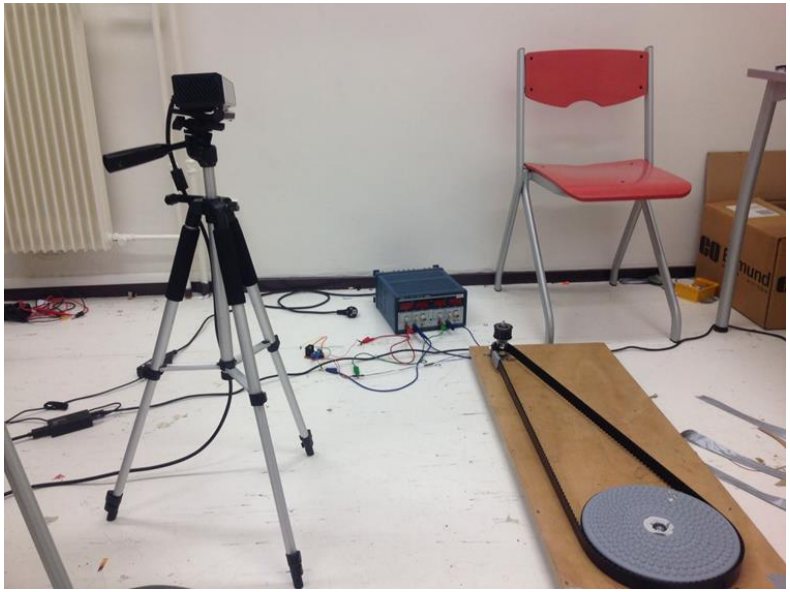






Turning Table (Serial Communication)

- Using Qtserialport class to communicate Qt with Arduino board.
- Arduino Sending PWM signal to control the DC motor.
- In the function on_scanButton_clicked(), while scan push button is pressed, it will send the logic 1 to the board, the motor will start to rotate.
- While the stop push button is pressed, it will send logic 0 to stop the motor.



Conclusion

What we have learned:

- Understanding how to interact with Kinect V2(configuring pro.file and linking the library).
- The basic idea and the pipeline of 3d point cloud reconstruction.
- How we manage and organize this project in a big group(10 people).
- How to use the PCL library functions and classes to fit in our project.
- How to use Qt creator to build GUI.