

Software specifications

**TOF CAMERA FUNCTION LIBRARY**

*Point Cloud Merge Function*

*Point Cloud to Polygon Conversion Function*

2020/08/26 Version: 1.1

Brycen Co., Ltd

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Revised parts and contents** | **Writer** |
| 1.0 | July 06, 2020 | Fist version | Nguyen Van Quoc Cuong |
|  | July 09, 2020 | Edit content | Nguyen Van Quoc Cuong |
|  | July 13, 2020 | Edit content | Nguyen Van Quoc Cuong |
|  | Aug. 2, 2020 | 1. Overall  1) Unified document format | K.Soma |
| 1.1 | Aug. 26, 2020 | 1. Edit software block diagram (1.4)  2. Change name of some function:  - **PolygonConversion** to **PointCloud2Polygon** (2.9)  - **ICP** to **MergePointCloud2** (2.10)  2. Add new function – **MergePointCloudN** (2.11) | Nguyen Van Quoc Cuong |
|  | Aug.31, 2020 | 1. Edit content in section 2.9, 2.10, 2.11 | Nguyen Van Quoc Cuong |
|  | Sep. 05, 2020 | 1. Change name of some function:  - **TriangleFace** to **IndexTriangle** (2.4)  - **MultiFaces** to **IndexTriangles** (2.5)  2. Add new function - **TFL\_LoadPolygon** (2.8)  3. Add Appendix page (4) | Nguyen Van Quoc Cuong |
|  | Sep. 09, 2020 | 1. Add new Appendix:  - **ICP Registration** (4.3)  - **VoxelGrid Filter** (4.4) | Nguyen Van Quoc Cuong |

**Contents**

**[1. Software Function Specifications:](#_Toc50556372)** [6](#_Toc50556372)

**[1.1. Operating environment](#_Toc50556373)** [6](#_Toc50556373)

**[1.2. Development environment and language](#_Toc50556374)** [6](#_Toc50556374)

**[1.3. List of software function](#_Toc50556375)** [6](#_Toc50556375)

**[1.4. Software Block Diagram](#_Toc50556376)** [7](#_Toc50556376)

[1.4.1. Software block explanation 7](#_Toc50556377)

[1.4.2. List of software modules 7](#_Toc50556378)

**[2. Library API Specification](#_Toc50556379)** [8](#_Toc50556379)

**[2.1. Overview](#_Toc50556380)** [8](#_Toc50556380)

**[2.2. Supported type list](#_Toc50556381)** [8](#_Toc50556381)

**[2.3. enum: TFL::TFL\_RESULT](#_Toc50556382)** [9](#_Toc50556382)

**[2.4. Struct: TFL::IndexTriangle](#_Toc50556383)** [11](#_Toc50556383)

**[2.5. Typedef: TFL::IndexTriangles](#_Toc50556384)** [12](#_Toc50556384)

**[2.6. Typedef: TFL::TFL\_PointCloud](#_Toc50556385)** [12](#_Toc50556385)

**[2.7. Struct: TFL::TFL\_Polygon](#_Toc50556386)** [13](#_Toc50556386)

**[2.8. Function: TFL::LoadPolygon](#_Toc50556387)** [13](#_Toc50556387)

**[2.9. Function: TFL::SavePolygon](#_Toc50556388)** [14](#_Toc50556388)

**[2.10. Function: TFL:: PointCloud2Polygon](#_Toc50556389)** [15](#_Toc50556389)

**[2.11. Function: TFL::MergePointCloud2](#_Toc50556390)** [15](#_Toc50556390)

[2.11.1. MergePointCloud2 function block diagram 17](#_Toc50556391)

[2.11.2. MergePointCloud2 function block diagram explanation 17](#_Toc50556392)

**[2.12. Function: TFL::MergePointCloudN](#_Toc50556393)** [19](#_Toc50556393)

[2.12.1. MergePointCloudN function block diagram 20](#_Toc50556394)

[2.12.2 MergePointCloudN block diagram explanation 20](#_Toc50556395)

**[3. External API](#_Toc50556396)** [21](#_Toc50556396)

**[4. Appendix](#_Toc50556397)** [22](#_Toc50556397)

**[4.1. Fast Point Feature Histogram (FPFH)](#_Toc50556398)** [22](#_Toc50556398)

**[4.2. Local Registration](#_Toc50556399)** [24](#_Toc50556399)

**[4.3. ICP Registration](#_Toc50556400)** [25](#_Toc50556400)

**[4.4. VoxelGrid Filter](#_Toc50556401)** [26](#_Toc50556401)

[4.4.1. Voxel 26](#_Toc50556402)

[4.4.2. VoxelGrid 27](#_Toc50556403)

[4.4.3. Downsampling 28](#_Toc50556404)

**Introduce**

This document is a software specification of ToF camera function library (TFLIB) limited to “*Point Cloud Merging”* and “*Point Cloud to Polygon” function*.

It is written for the purpose of clarifying the following:

1. Software Function Specifications.
2. Library API (Application Interface) Specifications.

# **1. Software Function Specifications:**

## **1.1. Operating environment**

The operating environment of this software library is as follows.

|  |  |
| --- | --- |
| **Item** | **Condition** |
| Platform | Windows 10 64bit |
| C/C++ runtime environment | Microsoft C++ runtime library 2019 |

Table 1 Operating environment

## **1.2. Development environment and language**

The development environment of this software library is as follows.

|  |  |
| --- | --- |
| **Item** | **Condition** |
| Platform | Windows 10 64bit |
| IDE | Visual Studio 2019 Pro |
| Build tools | MS Build version 16.4 |
| Programming language | C++ |

Table 2 Development environment

## **1.3. List of software function**

This software library realizes the following functions:

1. Merge pairs of point cloud data using ICP algorithm and return a new point cloud data to create 3D model.
2. Convert point cloud data to polygon and return a new polygon data.

## **1.4. Software Block Diagram**

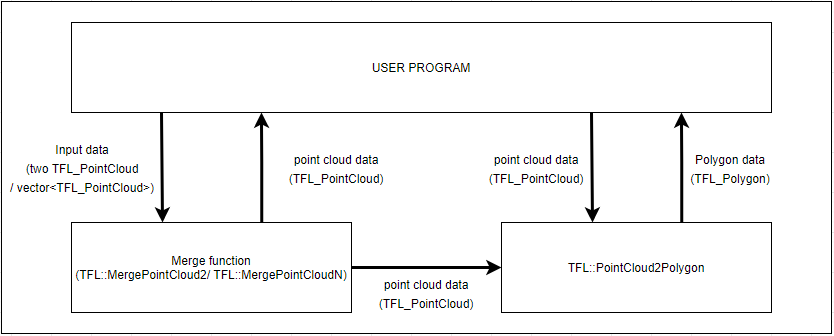


Figure 1 Software block diagram

### **1.4.1. Software block explanation**

|  |  |  |
| --- | --- | --- |
| No | Block | Overview |
| 1 | TFL::MergePointCloud2 | Merge pairs of point clouds using ICP algorithm |
| 2 | TFL::MergePointCloudN | Merge set of point clouds using ICP (Iterative Closest Point) algorithm |
| 3 | TFL::PointCloud2Polygon | Convert point cloud to polygon |

Table 3 Block explanation

### **1.4.2. List of software modules**

|  |  |  |
| --- | --- | --- |
| No | File name | Overview |
| 1 | tflib.h | Declaration file for all prototypes of the software Refer to both external module and this module. |
| 2 | tfl3DModeling.cpp | Source file of this software (C++) |

Table 4 Software modules

# **2. Library API Specification**

## **2.1. Overview**

1. Executable binary format: dll (64bit) for Windows
2. DLL name: tflib.dll
3. DLL header file name: tflib.h
4. Namespace: TFL

## **2.2. Supported type list**

Supported C++ types (class, structure…) provided by this library as below:

|  |  |  |  |
| --- | --- | --- | --- |
| No | Name | Type | Content |
| 1 | TFL::TFL\_RESULT | enum | Execution result of functions (normal and error code)  \* List up new enum value added by this specification. |
| 2 | TFL::TFL\_IndexTriangle | Struct | A structure that represents for index of 3 vertices of a triangle face (each vertex (point) is array index of point cloud data) |
| 3 | TFL::TFL\_IndexTriangles | Typedef | A std::vector array that stores set of multiple IndexTriangle |
| 4 | TFL::TFL\_PointCloud | Typedef | A std::vector array that stores point cloud data (variable size)  \* Since this item already exists, details are not described in this specification. |
| 5 | TFL::TFL\_Polygon | Typedef | Std::vector array that stores:  - points coordinate  - IndexTriangles data |
| 6 | TFL::LoadPLY | Function | Load point cloud data from PLY file  \* Since this item already exists, details are not described in this specification. |
| 7 | TFL::SavePLY | Function | Save point cloud data to PLY file  \* Since this item already exists, details are not described in this specification. |
| 8 | TFL::LoadPolygon | Function | Load polygon (TFL\_Polygon) from PLY file (include Point Cloud data and IndexTriangles data) |
| 9 | TFL::SavePolygon | Function | Save polygon (TFL\_Polygon) to PLY file |
| 10 | TFL::PointCloud2Polygon | Function | Convert point cloud to polygon |
| 11 | TFL::MergePointCloud2 | Function | Merge pairs of point clouds using ICP algorithm |
| 12 | TFL::MergePointCloudN | Function | Merge set of point clouds using ICP algorithm |

Table 5 Supported type

## **2.3. enum: TFL::TFL\_RESULT**

It is indicating execution result of this library API, and member list of the enum are below.

* Definition file: tflib.h
* Existing name space: TFL

|  |  |  |  |
| --- | --- | --- | --- |
| **TFL::TFL\_RESULT** | | | |
| **No** | **Valiable name** | **value** | **Summary** |
| 1 | TFL\_OK | 0 | Success (none of error) |
| 2 | TFL\_ERROR | 1 | Error of OS level (call GetLastError() to get OS error code) |
| 3 | TFL\_EXPIRED | 2 | Exceeded expiry of date |
| 4 | TFL\_CHANGED | 3 | Found invalid registry of expiry date. |
| 5 | TFL\_ZERODAY | 4 | Parameter for CheckExpiryDate() is zero. (Commonly impossible errors) |
| 6 | TFL\_PARAM1 | 11 | Invalid 1st parameter specification  (For details, refer to the corresponding API specifications) |
| 7 | TFL\_PARAM2 | 12 | Invalid 2nd parameter specification  (ditto) |
| 8 | TFL\_PARAM3 | 13 | Invalid 3rd parameter specification  (ditto) |
| 9 | TFL\_PARAM4 | 14 | Invalid 4th parameter specification  (ditto) |
| 10 | TFL\_PARAM5 | 15 | Invalid 5th parameter specification  (ditto) |
| 11 | TFL\_PARAM6 | 16 | Invalid 6th parameter specification  (ditto) |
| 12 | TFL\_PARAM7 | 17 | Invalid 7th parameter specification  (ditto) |
| 13 | TFL\_PARAM8 | 18 | Invalid 8th parameter specification  (ditto) |
| 14 | TFL\_PARAM9 | 19 | Invalid 9th parameter specification  (ditto) |
| 15 | TFL\_PARAM10 | 20 | Invalid 10th parameter specification  (ditto) |
| 16 | TFL\_PARAM11 | 21 | Invalid 11th parameter specification  (ditto) |
| 17 | TFL\_PARAM12 | 22 | Invalid 12th parameter specification  (ditto) |

Table 6 TFL\_Result parameters

## **2.4. Struct: TFL::IndexTriangle**

A closed set of edges, in which a triangle has three edges and three vertices (3 points). Therefore, TFL::IndexTriangle is a structure that represents for 3 vertices of a triangle ( value of each vertex is not value of point (x, y, z), but the index of points (position) in point cloud data (TFL\_PointXYZ array))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TFL::TFL\_IndexTriangle** | | | | |
| **No** | **Variables name** | **Type** | **Access** | **Content** |
| 1 | v1 | uint\_32 | Public | Index of first vertex of triangle face |
| 2 | v2 | uint\_32 | Public | Index of second vertex of triangle face |
| 3 | v3 | uint\_32 | Public | Index of third vertex of triangle face |

Table 7 List of TFL\_IndexTriangle elements

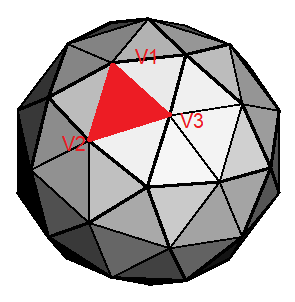
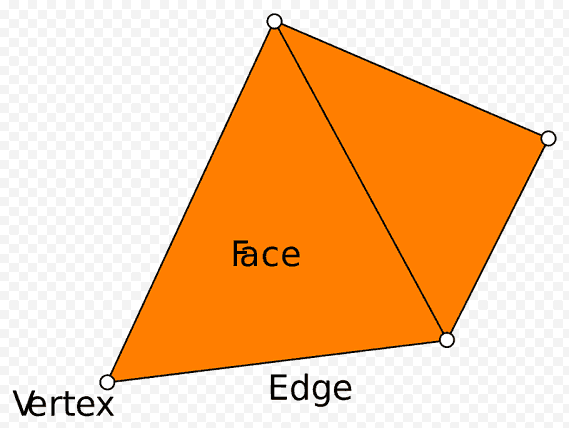


Figure 2 TFL\_IndexTriangle explanation

## **2.5. Typedef: TFL::IndexTriangles**

A IndexTriangles is a coplanar set of Indextriangle.

|  |  |
| --- | --- |
| **TFL::TFL\_IndexTriangles** | |
| Type | typedef TFL\_DLL std::vector<TFL\_IndexTriangle> TFL\_IndexTriangles |

Table 8 Typedef: TF::TFL\_IndexTriangles

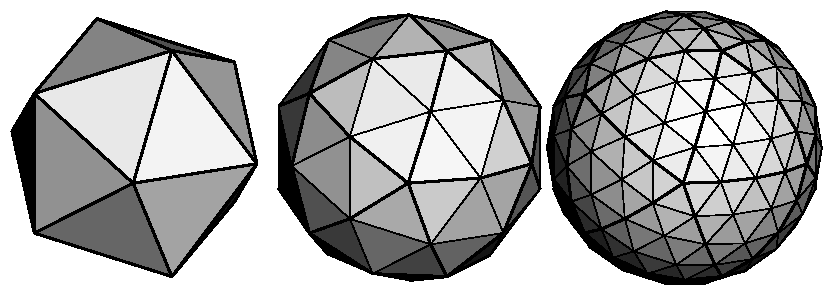


Figure 3 IndexTriangles is set of IndexTriangle

## **2.6. Typedef: TFL::TFL\_PointCloud**

Point cloud data that holds variable size of a point data. It is defined as follows.

|  |  |
| --- | --- |
| **TFL::TFL\_PointCloud** | |
| Type | typedef TFL\_DLL vector<TFL\_PointXYZ> TFL\_PointCloud |

Table 9 Typedef: TFL::TFL\_PointCloud

## **2.7. Struct: TFL::TFL\_Polygon**

Point cloud data that holds variable size of points coordinate and IndexTriangles (set of IndexTriangle). It is defined as follows

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TFL::TFL\_Polygon** | | | | |
| **No** | **Variables name** | **Type** | **Access** | **Content** |
| 1 | PointCloud | TFL::TFL\_PointCloud | Public | Point Cloud data |
| 2 | IndexTriangles | TFL::TFL\_IndexTriangles | Public | IndexTriangles data |

Table 10 List of TFL\_Polygon elements

## **2.8. Function: TFL::LoadPolygon**

|  |  |  |  |
| --- | --- | --- | --- |
| Function | LoadPolygon | | |
| Overview | Load Polygon data from PLY file | | |
| Prototype | LoadPolygon(  wchar\_t\* filename  const TFL::TFL\_Polygon &polygon) | | |
| Return | enum TFL::TFL\_Result | | |
|  | Name | I/O | Contents |
| Argument | *&filename* | IN | File name path |
| *&polygon* | IN | Polygon (TFL\_Polygon) data |
| Note | When *&filename content* is illegal (content is NULL or wrong syntax), the return value will be **TFL\_Result::TFL\_PARAM1** | | |

Table 11 Load Point Cloud function

## **2.9. Function: TFL::SavePolygon**

|  |  |  |  |
| --- | --- | --- | --- |
| Function | SavePolygon | | |
| Overview | Save Polygon data to PLY file | | |
| Prototype | SavePolygon(  wchar\_t\* filename  const TFL::TFL\_Polygon &polygon) | | |
| Return | enum TFL::TFL\_Result | | |
|  | Name | I/O | Contents |
| Argument | *&filename* | IN | File name path |
| *&polygon* | IN | Polygon (TFL\_Polygon) data |
| Note | When *&polygon* is NULL, the return value will be **TFL\_Result::TFL\_PARAM1** | | |

Table 12 Save Point Cloud function

## **2.10. Function: TFL:: PointCloud2Polygon**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **PointCloud2Polygon** | | |
| **Overview** | Convert point cloud data to polygon | | |
| **Prototype** | PointCLoud2Polygon(  const TLF::TFL\_PointCloud &pointcloud,  float voxel\_size,  const TFL::TFL\_Polygon &polygon) | | |
| **Return** | enum TFL::TFL\_Result | | |
|  | Name | I/O | Contents |
| **Argument** | *&pointcloud* | IN | Point Cloud data |
| *voxel\_size* | IN | Before conversion process, this function will do the down sampling for input point cloud data (*&pointcloud*) (also using the voxelgrid filter).  And the size of each 3D voxel grid edge is determined by this parameter.  \* More details in **4.4. VoxelGrid Filter** |
| *&polygon* | OUT | Polygon data  Result of PointCloud2Polygon process |
| Note | When *&pointcloud* is NULL, the return value will be **TFL\_Result::TFL\_PARAM1** | | |

Table 13 PointCloud2Polygon function

## **2.11. Function: TFL::MergePointCloud2**

|  |  |  |  |
| --- | --- | --- | --- |
| Function | MergePointCloud2 | | |
| Overview | Merge pairs of point cloud data using ICP (Iterative Closest Point) algorithm | | |
| Prototype | MergePointCloud2 (  const TLF::TFL\_PointCloud &source,  const TLF::TFL\_PointCloud &target,  float SizeVoxel\_preprocess  float SizeVoxel\_postprocess  const TFL\_PointCloud &result) | | |
| Return | enum TFL::TFL\_Result | | |
|  | Name | I/O | Contents |
| **Argument** | *&source* | IN | Point Cloud data |
| *&target* | IN | Point Cloud data |
| SizeVoxel\_preprocess | IN | One of the steps of preprocessing is down sampling.  And down sampling is done using the VoxelGrid filter (the Voxelgrid filter creates a 3D voxel grid to do the down sampling (Like the down sampling function I showed earlier).  The size of each edge of the 3D voxel grid is determined by this parameter.  \* More details in **4.4. VoxelGrid Filter** |
| *SizeVoxel\_postprocess* | IN | After merging, this function will do the down sampling for the medged pointcloud data again (also using the voxelgrid filter).  And the size of each 3D voxel grid edge is determined by this parameter.  \* More details in **4.4. VoxelGrid Filter** |
| *Result* | OUT | Point Cloud data  Result of MergePointCloud2 process |
| Note | When *&source* or *&target* are NULL, the return value will be **TFL\_Result::TFL\_PARAM1** | | |

Table 14 MergePointCloud2 function

### **2.11.1. MergePointCloud2 function block diagram**

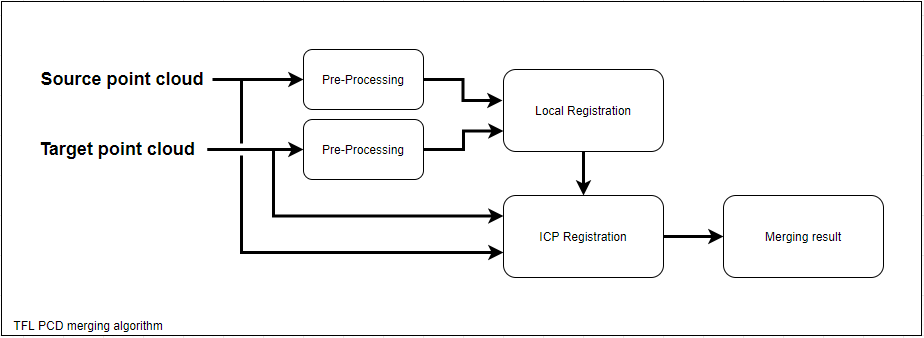


Figure 4 MergePointCloud2 block diagram

### **2.11.2. MergePointCloud2 function block diagram explanation**

|  |  |  |
| --- | --- | --- |
| **No** | **Block** | **Explanation** |
| 1 | Source point cloud | Source point cloud and target point cloud are input of MergePointCloud2() algorithm.  - Source point cloud: will be moved depending on target point cloud.  - Target point cloud: is used to orient, it stands still and will be the target for the source point to find the correct direction to move. |
| 2 | Target point cloud |
| 3 | Pre-processing | We down sample the point cloud, estimate normal, then compute a FPFH feature for each point.  The FPFH feature is a 33-dimensional vector that describes the local geometric property of a point.  A nearest neighbor query in the 33-dimensinal space can return points with similar local geometric structures.  \* More details in **4.1. Fast Point Feature Histogram (FPFH)** |
| 4 | Local Registration | Pre-processing result is input of local registration.  We use RANSAC for local registration, in each Ransac iteration, **ransac\_n** random points are picked from the source point cloud.  Their corresponding points in the target point cloud are detected by querying the nearest neighbor in the 33-dimensional FPFH feature space.  A pruning step takes fast pruning algorithms to quickly reject false matches early.  \* More details in **4.1. Fast Point Feature Histogram (FPFH)** and **4.2. Local Registration** |
| 5 | ICP Registration | For performance reason, the Local registration is only performed on a heavily down-sampled point cloud.  The result is also not tight. We must use ICP registration (point to plan) (this step) to further refine the alignment.  \* More details in **4.3. ICP Registration** |
| 6 | Merging Result | The MergePointCloud2() result is source point cloud after moving (transform) based on data from local registration and icp registration. |

Table 15 Explanation of MergePointCloud2 block

## **2.12. Function: TFL::MergePointCloudN**

|  |  |  |  |
| --- | --- | --- | --- |
| Function | MergePointCloudN | | |
| Overview | Merge set of point clouds using ICP (Iterative Closest Point) algorithm | | |
| Prototype | MergePointCloudN(  Std::vector<TFL::TFL\_PointCloud &vector>,  float SizeVoxel\_preprocess,  float SizeVoxel\_postprocess,  const TFL::TFL\_PointCloud &result) | | |
| Return | enum TFL::TFL\_Result | | |
|  | Name | I/O | Contents |
| Argument | *&vector* | IN | Set of Point Cloud data |
| SizeVoxel\_preprocess | IN | Voxel Grid edge size for pre-processing (meter) |
| SizeVoxel\_postprocess | IN | Voxel Grid edge size (meter) |
| *Result* | OUT | Point Cloud data  Result of MergePointCloudN process |
| Note | When *&vector members (TFL\_PointCloud)* are NULL, the return value will be **TFL\_Result::TFL\_PARAM1** | | |

Table 16 MergePointCloudN function

### **2.12.1. MergePointCloudN function block diagram**

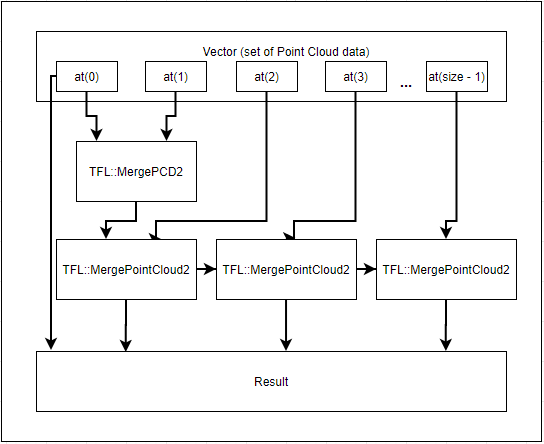


Figure 5 MergePointCloudN block diagram

2.12.2 MergePointCloudN block diagram explanation

|  |  |  |
| --- | --- | --- |
| **No** | **Block** | **Explanation** |
| 1 | Vector | Set of point cloud data  **Std::vector <TFL::TFL\_PointCloud>** |
| 2 | at(0), at(1),… at(size – 1) | Position of frames (Point cloud data) which stored in vector  The number of frames is **vector.size()** |
| 3 | TFL::MergePointCloud2 | MergePointCloud2() process between 2 frames and return a point cloud data (TFL::TFL\_PointCloud) |
| 4 | Result | Result of MergePointCloudN function |

Table 17 Explanation of MergePointCloudN block

3. External API

|  |  |  |
| --- | --- | --- |
| **External API** | | **Reference Link** |
| PCL 1.10.1 | | <https://github.com/PointCloudLibrary/pcl/tree/pcl-1.10.1> |
| Open3D 1.10.0 | | <https://github.com/intel-isl/Open3D/tree/v0.10.0> |
| 3Rd party | For PCL | **Boost 1.72.0** [https://gist.github.com/UnaNancyOwen/d879a41710e9c05025f8#file-boost1-72-0-md](https://gist.github.com/UnaNancyOwen/d879a41710e9c05025f8" \l "file-boost1-72-0-md)  **Eigen 3.3.7** [https://gist.github.com/UnaNancyOwen/08e5a9b8f5979ed0077c#file-eigen3-3-7-md](https://gist.github.com/UnaNancyOwen/08e5a9b8f5979ed0077c" \l "file-eigen3-3-7-md)  **FLANN 1.9.1** [https://gist.github.com/UnaNancyOwen/1e3fced09e4430ad0b7b#file-flann1-9-1-md](https://gist.github.com/UnaNancyOwen/1e3fced09e4430ad0b7b" \l "file-flann1-9-1-md)  **QHull 2019.1** [https://gist.github.com/UnaNancyOwen/ffbe6b05ee0101620d3c#file-qhull-2019-1-md](https://gist.github.com/UnaNancyOwen/ffbe6b05ee0101620d3c" \l "file-qhull-2019-1-md)  **VTK 8.2.0** [https://gist.github.com/UnaNancyOwen/9d16060714ba9b28f90e#file-vtk8-2-0-md](https://gist.github.com/UnaNancyOwen/9d16060714ba9b28f90e" \l "file-vtk8-2-0-md)  **OpenNI 2.2.0.33 Beta** <http://structure.io/openni> |
| For Open3D | 3rd party libraries are free (even for commercial purposes) and there are no special requests from the software publishers.  <https://github.com/intel-isl/Open3D/tree/master/3rdparty> |

Table 18 External API

4. Appendix

## **4.1. Fast Point Feature Histogram (FPFH)**

Point Feature Histogram (PFH) encode the geometric properties of the k nearest neighbors of a point by using the average curvature of the multidimensional histogram around a point. This high-dimensional space provides some useful feature representations, while being invariant to the 6-position, and can cope with different sampling densities and noise levels of neighbors.

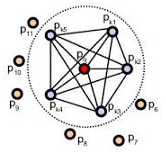


Figure 6 The area of influence of a PFH

The center point and all its neighboring points inside the sphere with a radius of r are connected to each other to form a network. The final PFH geometric feature will be a histogram obtained by calculating the change relationship between all pairs of points in the neighborhood.

In order to calculate the relative difference between the two, given two points, **Ps** and **Pt**, and their associated normals **ns** and **nt**, we define a fixed coordinate system at a certain point.

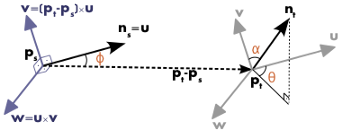


Figure 7 Ps and Pt, and their associated normals ns and nt

Using the uvw coordinate system above, the difference between the **ns** normal and the **nt** normal can be resolved into the following 3 angle differences

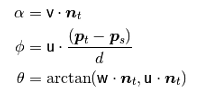


Figure 8 Formula to calculate 3 angle differences

These variables, together with the Euclidean distance between the points, are saved, and then binned to a histogram when all pairs have been computed.

The final descriptor is the concatenation of the histograms of each variable.

Fast Point Feature Histogram (FPFH) is an extension of PFH, it shares most of the characteristics and principles with this geometric feature of PFH.

The input of FPFH is also a point cloud with normal information, and the output is a histogram that can reflect the characteristics of the neighborhood around each point. But unlike PFH, some simplification and optimization measures will be taken to speed up the calculation of FPFH.

The following specifically describes how FPFH makes calculations faster through simplification and optimization.

First, for each point, use a method similar to PFH to calculate the triples and obtain a simplified Point Feature Histogram referred as SPFH.

Use a weighted neighboring SPFH to calculate final value of the histogram as follows:

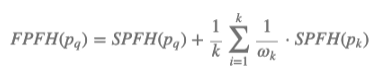


Figure 9 Calculate final value of the histogram

Where the weights depend on the center point and a neighbor point at a given distance metric space. Of course, other measurement methods can also be used here to set this weight.

## **4.2. Local Registration**

For performance reason, the Local registration is only performed on a heavily down-sampled (In Pre-processing step) point cloud.

Following figure is result of Local Registration process. We can easily to see that:

- The density of points is low.

- The result is also not tight

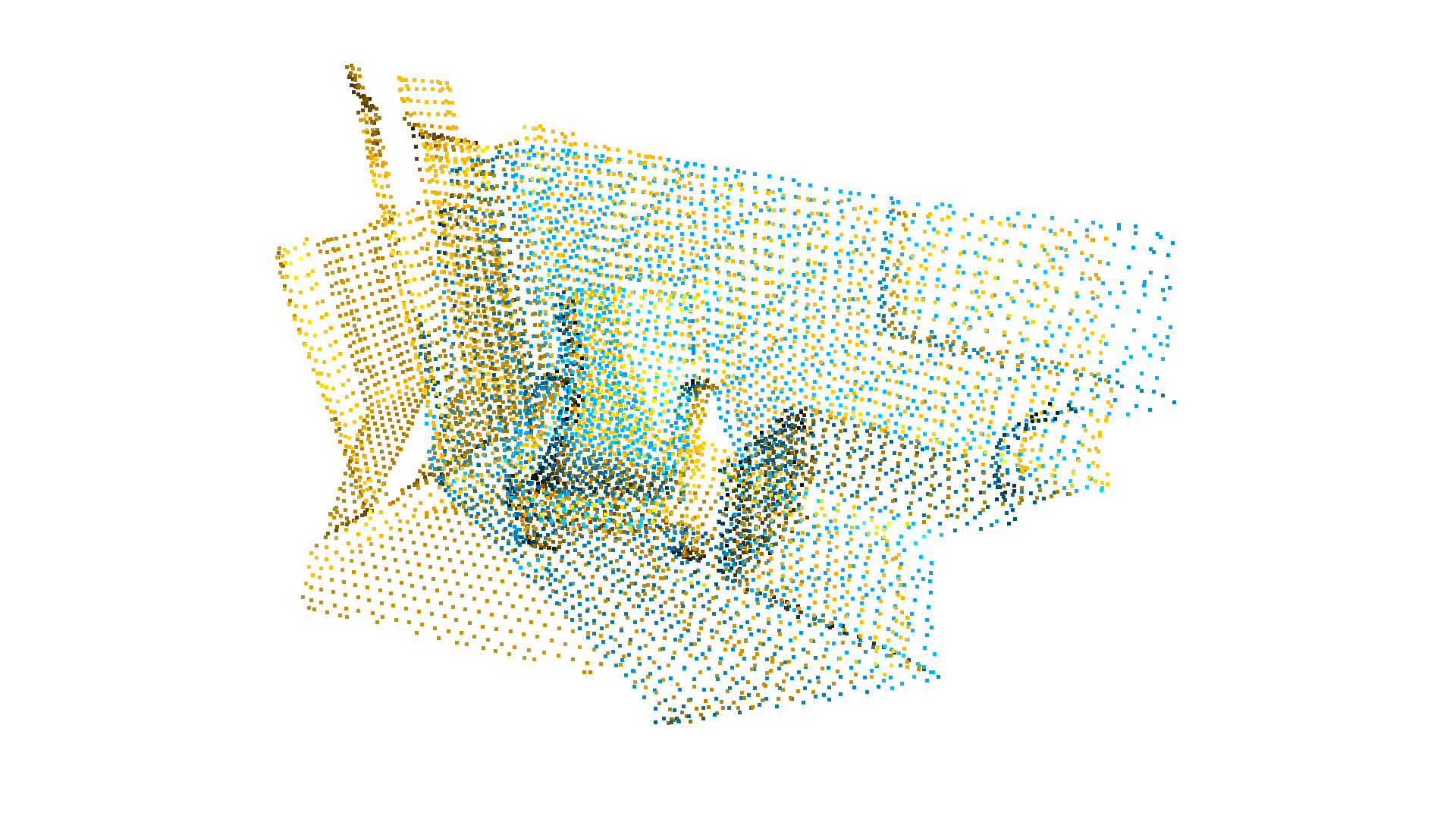


Figure 10 Result of Local Registration

We must use Point-to-Plane ICP in next step (ICP Registration) to further refine the alignment.

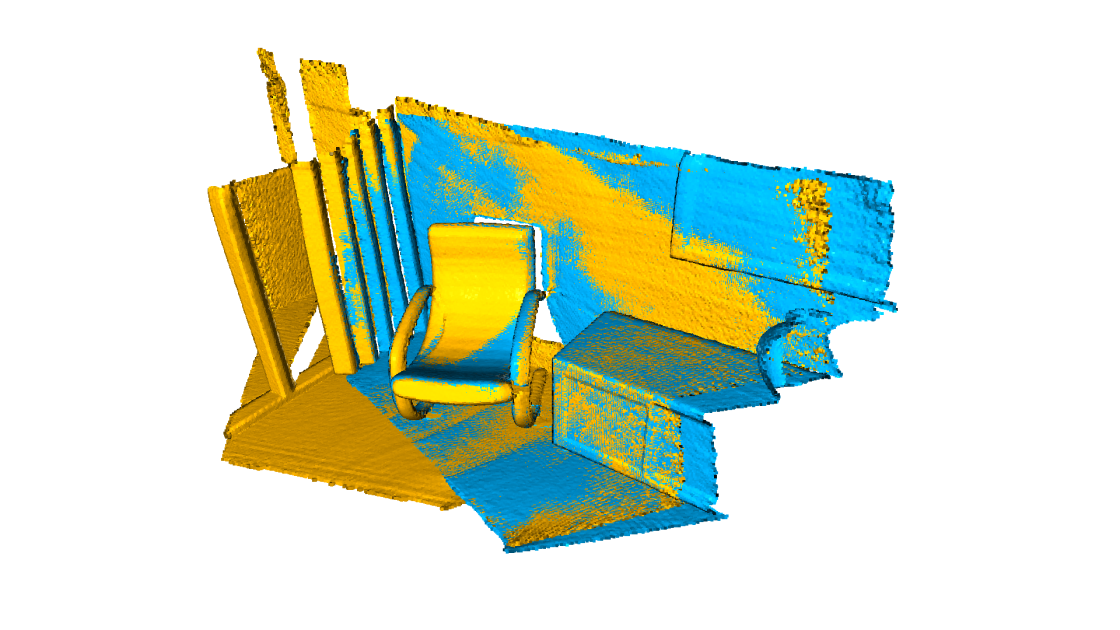


Figure 11 Result of Global Registration

## **4.3. ICP Registration**

This tutorial demonstrates the ICP (Iterative Closest Point) registration algorithm. It has been a mainstay of geometric registration in both research and industry for many years. The input are two point clouds and an initial transformation that roughly aligns the source point cloud to the target point cloud. The output is a refined transformation that tightly aligns the two point clouds. A helper function draw\_registration\_result visualizes the alignment during the registration process. In this tutorial, we show two ICP variants, the ***point-to-point ICP*** and the ***point-to-plane ICP***.In the scope of this specification, I will explain only ***Point-to-Plane***.

Given a source surface and a destination surface, each iteration of the ICP algorithm first establishes a set of pair-correspondences between points in the source surface and points in the destination surfaces. For example, for each point on the source surface, the nearest point on the destination surface is chosen as its correspondence for other approaches to find point correspondences). The output of an ICP iteration is a 3D rigid-body transformation M that transforms the source points such that the total error between the corresponding points, under a certain chosen error metric, is minimal. When the point-to-plane error metric is used, the object of minimization is the sum of the squared distance between each source point and the tangent plane at its corresponding destination point (see *Figure 13*). More specifically, if ***s­i = (six , siy , siz , 1)T*** is a source point, ***di = (dix , diy , diz , 1)T*** is the corresponding destination point, and ***ni = (nix , niy , niz , 0)T*** is the unit normal vector at di , then the goal of each ICP iteration is to find Mopt such that:



Figure 12 Formula for calculating Mopt

Where M and Mopt are 4×4 3D rigid-body transformation matrices.

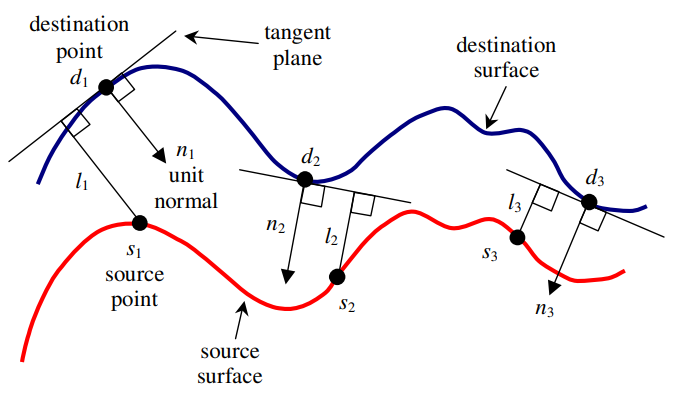


Figure 13 Point-to-plane error between two surfaces

A 3D rigid-body transformation **M** is composed of a rotation matrix ***R****(α, β, γ)* and a translation matrix *T(tx , ty , tz)*, i.e.

***M = T( tx, ty, tz) ⋅ R(α, β,γ )*** *(2)*

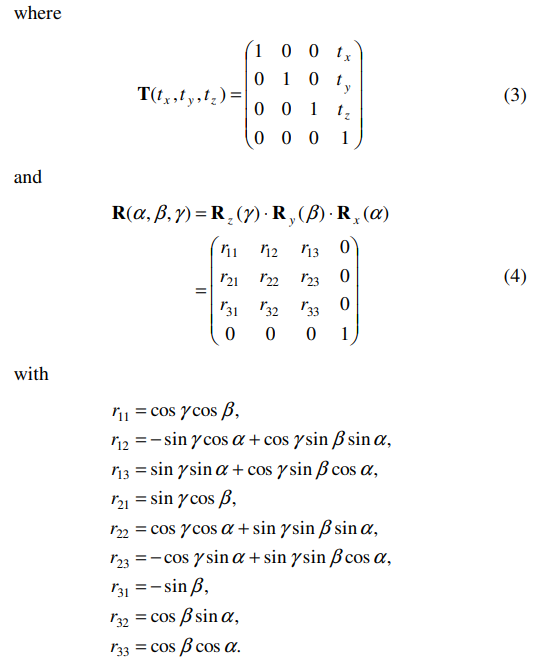


Figure 14 Matrix *R(α, β, γ)* and a translation matrix *T(tx , ty , tz)*

*Rx(α), Ry(β) and Rz(γ)* are rotations of *α, β,* and *γ* radians about the x-axis, y-axis and z-axis, respectively. Equation (1) is essentially a least-squares optimization problem, and solving it requires the determination of only the values of the six parameters *α, β, γ, tx, ty*, and *tz*. However, since *α, β*, and *γ* are arguments of nonlinear trigonometric functions in the rotation matrix **R**, efficient linear least-squares techniques cannot be applied to obtain the solution. In the next section, we present how this nonlinear least-squares problem can be approximated by a linear one, so that a linear least-squares technique can be applied.

## **4.4. VoxelGrid Filter**

### **4.4.1. Voxel**

Voxel is a terminology in 3D Graphics. A voxel represents a value on a regular grid in three-dimensional space. As with pixels in a 2D bitmap, voxels themselves do not typically have their position (i.e. coordinates) explicitly encoded with their values. Instead, rendering systems infer the position of a voxel based upon its position relative to other voxels (i.e., its position in the data structure that makes up a single volumetric image).

In contrast to pixels and voxels, polygons are often explicitly represented by the coordinates of their vertices (as points). A direct consequence of this difference is that polygons can efficiently represent simple 3D structures with much empty or homogeneously filled space, while voxels excel at representing regularly sampled spaces that are non-homogeneously filled.

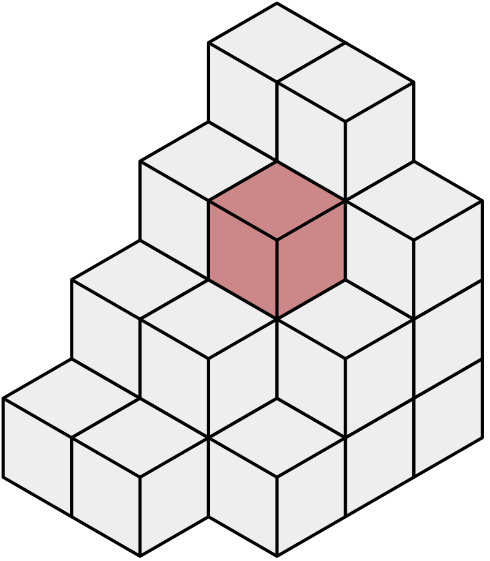


Figure 15 A series of voxels, with a single voxel shaded

### **4.4.2. VoxelGrid**

VoxelGrid is a set of voxel (3D boxes in space).

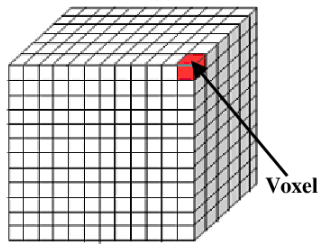
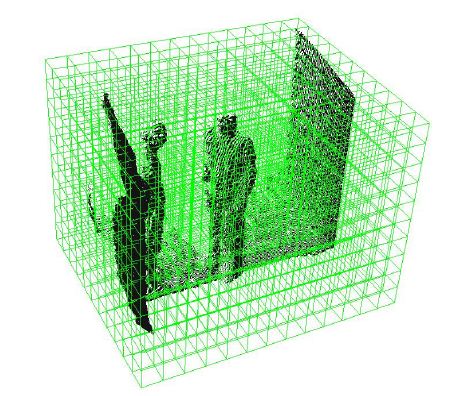


Figure 16 Voxel Grid is a set of 3D boxes (voxels) in space

### **4.4.3. Downsampling**

For easy to imagine, I will explain how downsampling works in 2D VoxelGrid first.

In the following figure, I show operation of downsampling function over a 2D point cloud containing uniformly dispersed points. I divide each axis into 5 regions resulting in 25 voxels. The points are down-sampled by taking the centroid of the points within each voxel. The centroids are shown in red asterisks. The centers of the voxels are shown in green squares.

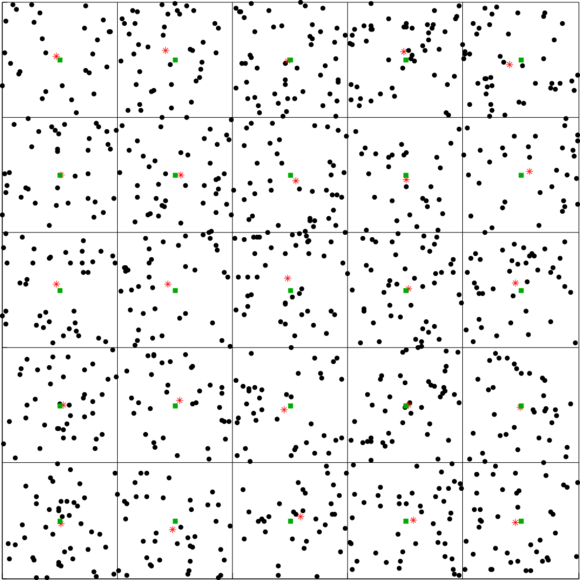


Figure 17 A 2D VoxelGrid with red stars representing the voxel centroids

The way downsampling works with 3D VoxelGrid is similar to 2D VoxelGrid.

From above explanation, It is easy to realize benefits of downsampling:

* Reduce number of points in Point Cloud data
* Arrange points according to responsible density