# Related technique

## PE

Core objective

2D --> 3D

### Applications

* Camera calibration
* Image orientation / alignment
* 3D recovery ( Stereo matching / LiDAR / Radar )
* 3D Model reconstruction

## CV

Core objective

2D --> 2D

2D --> semantic

### Applications

* Low-level
  + - Image enhancement
* Super-resolution
* Color Enhancement (HDR)
* Blur removal
* Smoke, haze, shadow removal
  + - Image stitching
* Panorama stitching
* Color blending
  + - Image-based rendering
* Light fields
* High-level
  + - Recognition
* Face recognition
* Class recognition (classification)
* Instance recognition (classification + segmentation)
  + - Context and scene understanding

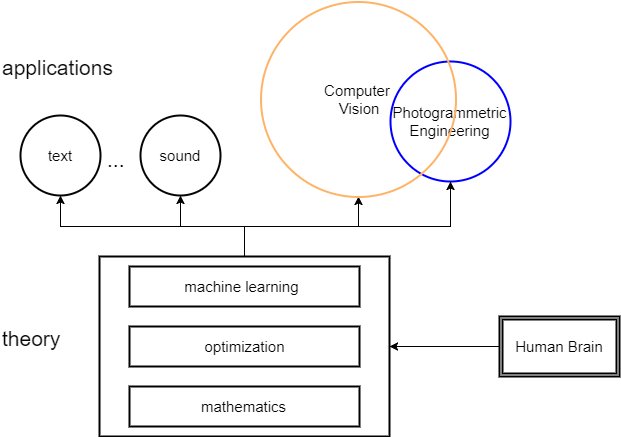
## ML

Core objective

Data --> semantic

* Feature selection
* Classifier design
* Optimized solution

## Relationship



# GIS

Raster / vector

Spatial analysis

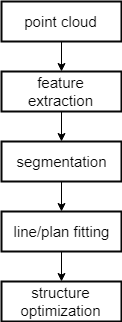
GEOS - Geometry Engine Open Source

<http://geos.refractions.net/ro/doxygen_docs/html/index.html>

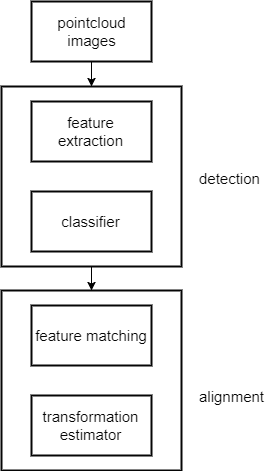
# Working flow

## Indoor Mapping

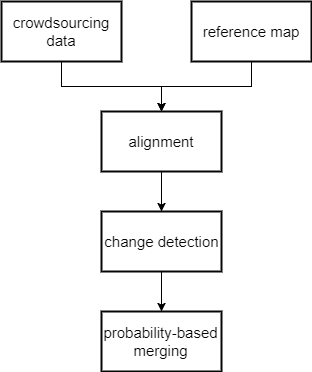
### Map Reconstruction



### Object Localization

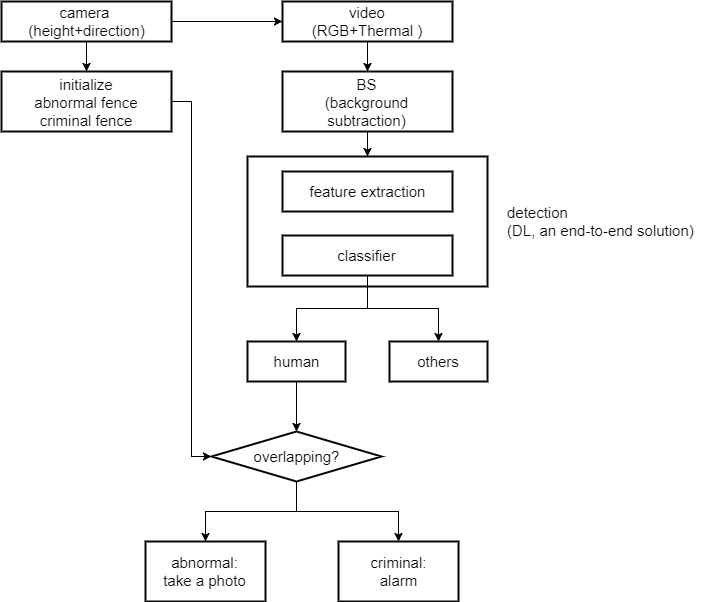


### Map Updating

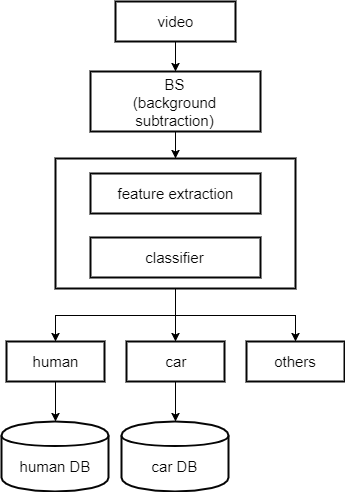


## Video Tracking

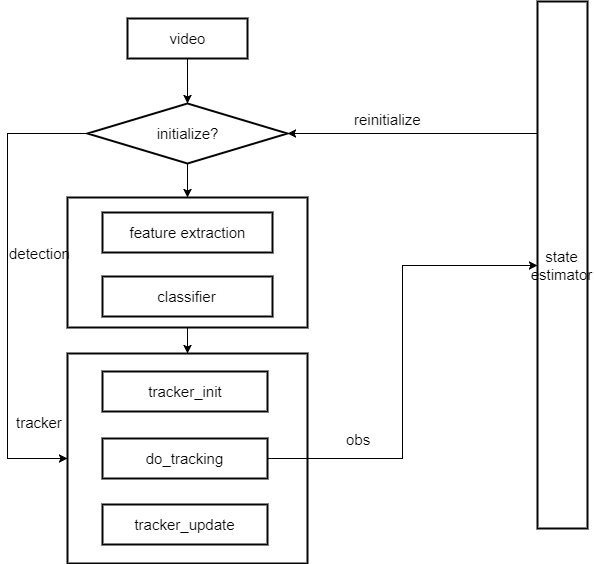
### Equipmen Surveillance



### Target Counting



### Pedestrian Tracking



# System Modules

## Core

### Functionality

Defining basic data structures and basic functions used by all other modules.

##### Iplstd： std中类型和函数的引用

Iplversion： 版本定义

Ipldef： ipl系统中的宏定义

Iplcore： ipl核心头文件

Iplstring： 字符串操作

Iplfiles： 文件操作

#### Iplstring

* 目前支持UTF-8编码，采用其他编码(unicode)的字符串在调用相关函数时，结果可能出错。
* 后续可考虑采用Boost.Locale机制实现本地化和编码转换。

#### Iplfiles

基于boost::filesystem 实现跨平台

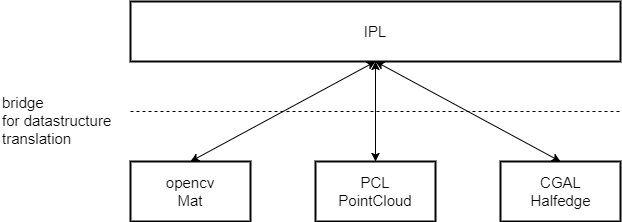
#### Iplcore

(1) IplException

需求：

* 能够接收3rdparty (openCV, PCL, CGAL) 抛出的异常，这个库的异常各不相同；
* User不需要知道调用了哪些库；
* User能够抛出自己的异常

1. data structure



## Classifier

**Input/output**

* Classify: feature vectors --> labels
* Training: feature vectors + labels --> classifier parameters

### Point cloud classifiers

(1) Classify wall/ceiling

/\*\* \brief classify pointcloud as wall or ceiling

\* \param[in] cloud the point cloud

\* \param[in] ver\_degTh the angle threshold for vertical points which measures the included angle to vertical direction

\* \param[in] hor\_degTh the angle threshold for horizontal points which measures the included angle to vertical direction

\* \param[out] indices, indices[0], wall; indices[1], ceiling; indices[2], others

\*/

//indices: indices[0], wall; indices[1], ceiling; indices[2], others

IPL\_EXPORTS int classifyWallCeiling(

iplPointCloud<iplPointXYZRGBNormal> \*cloud,

std::*vector*<std::*vector* <int> > &indices,

float ver\_degTh = 15.0,

float hor\_degTh = 80.0/\*, float hei\_interval = 0.1\*/);

Classical classifiers, do not support DL. As DL conbines feature selection and classifier training together. It needs a new framework.

* Random forest
* Adaboost
* SVM

## I/O

### Point cloud I/O

/\*\* \brief read pointcloud file

\* \param[in] file\_name

\* \param[out] cloud point cloud data

\* \param[in] offset the offset in the file where to expect the true header to begin.

\* \return

\* \* < 0 (-1) on error

\* \* == 0 on success

\*/

template<typename PointT>

int iplreadPointCloud(const std::*string* &file\_name,

iplPointCloud<PointT> &cloud, const int offset = 0);

/\*\* \brief write pointcloud file

\* \param[in] file\_name

\* \param[out] cloud point cloud data

\* \param[in] indices the indices of pointcloud that want to write out.

\*.\param[in] binary set to true if the file is to be written in a binary

\* format, false (default) for ASCII

\* \return

\* \* < 0 (-1) on error

\* \* == 0 on success

\*/

template<typename PointT>

int iplwritePointCloud(const std::*string* &file\_name,

const iplPointCloud<PointT> &cloud,

const std::*vector*<int> &indices,

const bool binary = false);

### Geometric model I/O

Use the 3rdparty libarary “Open Asset Import Library -- assimp”

CGAL Polyhedral off

Pcl PolygonMesh ply, obj

## Model Fitting

**Input/output**

* Segment: data + model type --> a batch of clusters + model parameters
* Fitting: data --> model parameters

### Geometric models

#### Plane

/\*\* \brief detect planes in a given point set

\* \param[in] cloud: point cloud

\* \param[in] indices: the indices of points which need to be processed

\*.\param[in] has\_normal: set to true if the point cloud has normal

\* \param[out] sorted\_cloud: the eRansac will sort the point cloud

\* \param[out] inliers: the indices of plane points, this indices is corresponding to sorted\_cloud

\*.\param[out] planeCoeffs: the coefficients of detected planes

\*.\param[in] minModelPts: the minimum number of points in one plane

\*.\param[in] epsTh: the threshold for plane fitting

\*.\param[in] epsConnective: the threshold for points connectivity

\*.\param[in] normalTh: the threshold for included angle of point normals, it is a cosine of angle

\* \return

\* \* < 0 (-1) on error

\* \* == 0 on success

\*/

template <typename PointT>

int detectPlanes\_eRansac(const iplPointCloud<PointT> &cloud,

std::*vector*<int> \*indices,

bool has\_normal,

iplPointCloud<PointT> &sorted\_cloud,

std::*vector*<std::*vector*<int> > &inliers,

std::*vector*<iplModelCoeffs> &planeCoeffs,

int minModelPts = 200,

double epsTh = 0.05,

double epsConnective = 0.2,

double normalTh = 0.9);

#### parameters I/O

/\*\* \brief read geometric model parameters from a file

\* \param[in] filename

\* \param[out] mType: the flag to designate the model type

\*.\param[out] mCoef: geometric model parameters

\* \return

\* \* < 0 (-1) on error

\* \* == 0 on success

\*/

IPL\_EXPORTS int loadGeoModelParams(const std::*string* &filename,

ipl::geoModel::geoModelType &mType,

ipl::iplModelCoeffs &mCoef);

/\*\* \brief save geometric model parameters in a file

\* \param[in] filename

\* \param[in] mType: the flag to designate the model type

\*.\param[in] mCoef: geometric model parameters

\* \return

\* \* < 0 (-1) on error

\* \* == 0 on success

\*/

IPL\_EXPORTS int saveGeoModelParams(const std::*string* &filename,

const ipl::geoModel::geoModelType mType,

const ipl::iplModelCoeffs &mCoef);

### Probabilistic models

## Geometry

* 2D geometry;
* 3D geometry;
* projective geomety

### Input/output

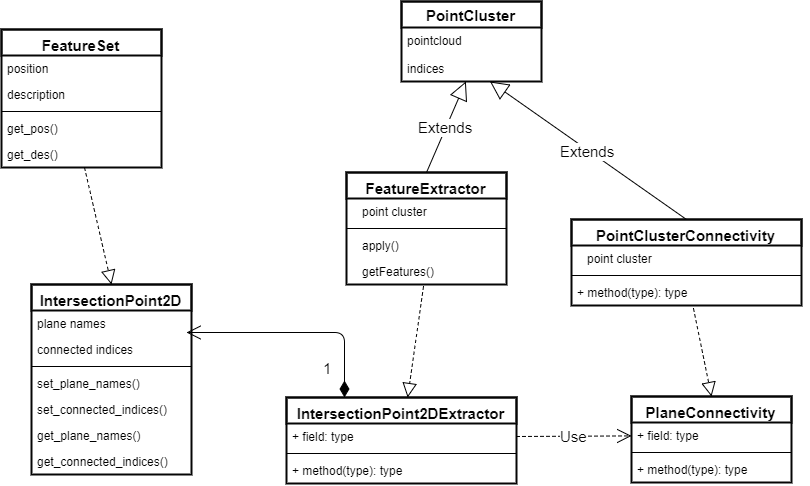
* Transform: Coordinate vectors --> coordinate vectors
* Estimate: Corresponding coordinate vectors --> transformation parameters

## Feature Extraction

**Input/output**

* Extract: Image/pointcloud --> feture vector/matrix
* Describe: image/pointcloud + coordinate --> descriptor vectors
* Image features
* Pointcloud features
* Point of interest
* Descriptors

### Point cloud



NOTE：FeatureExtractor的设计存在一些问题，不同派生类导出的feature类型不同，如何统一接口？

### image

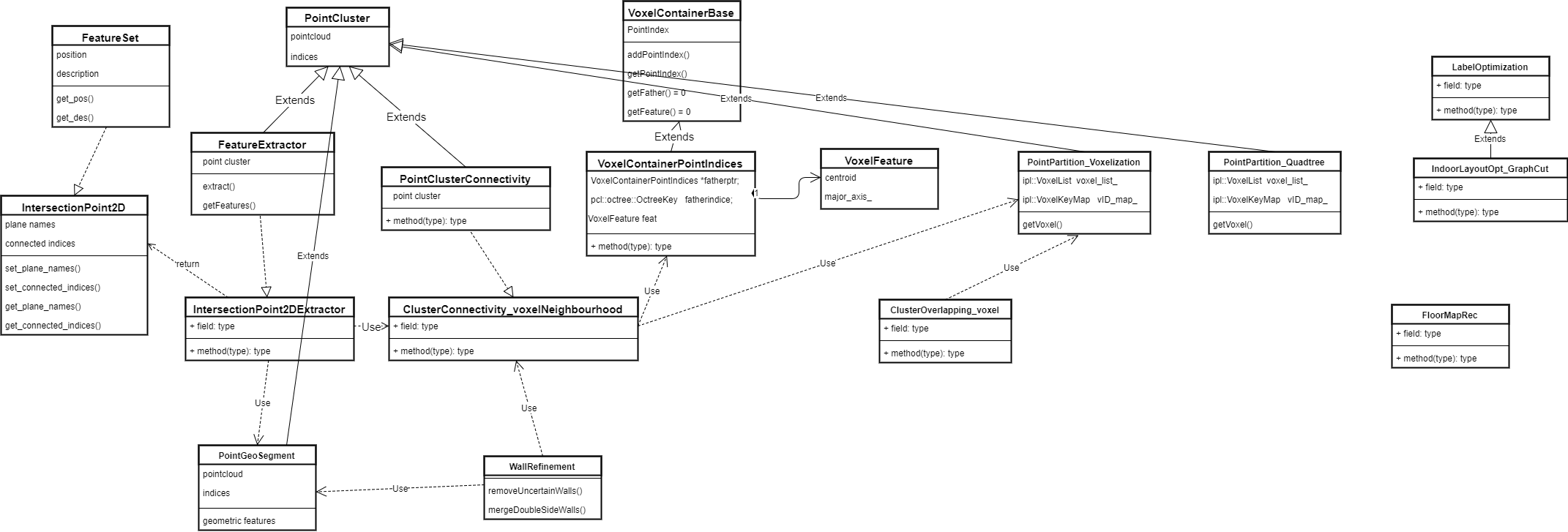
## Registration

**Input/output**

* Ref. Data, target data
* Transformation parameters

此模块的重构工作推迟到11月

## Optimization



* PGM (Probabilistic Graphical Model)
* DP (Dynamic Programming)
* GC (Graph cut)

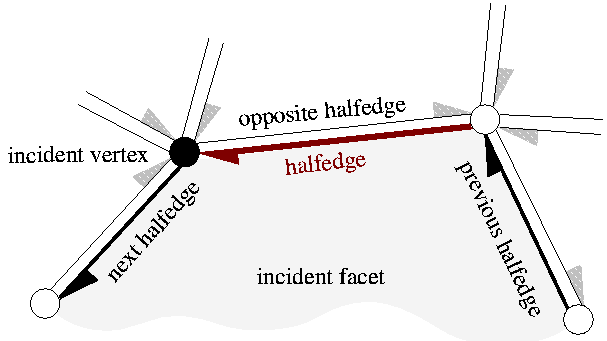
### Input/output

* Parameter optimization
* Label optimization

## Building Reconstruction

### Data structure

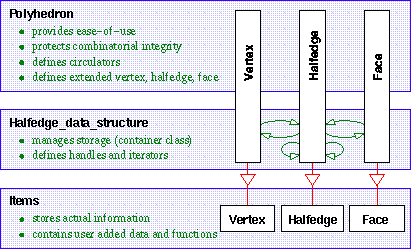
模型重建时，采用多面体Polyhedron来表示，能够表达语义，mesh更适用于作为一种交换格式。Polyhedron (subdivision)--> mesh



#### HalfedgeDS

* is an edge-centered data structure capable of maintaining incidence information of vertices, edges and faces. (orientable, two-dimentional surfaces embedded in arbitrary dimension)
* Each edge is decomposed into two halfedges with opposite orientations.
* One incident face and one incident vertex are stored in each halfedge.
* For each face and each vertex, one incident halfedge is stored.

#### Polyhedron\_3



A polyhedral surface **[Polyhedron\_3](https://doc.cgal.org/latest/Polyhedron/classCGAL_1_1Polyhedron__3.html" \o "A polyhedral surface Polyhedron_3 consists of vertices V, edges E, facets F and an incidence relation...)** consists of vertices V, edges E, facets F and an incidence relation on them.

* Vertices: represent points in 3d-space.
* Edges are straight line segments between two endpoints.
* Facets are planar polygons without holes defined by the circular sequence of halfedges along their boundary.

***border halfedges***

The halfedges along the boundary of a hole are called border halfedges and have no incident facet.

***border edge***

An edge is a border edge if one of its halfedges is a border halfedge.

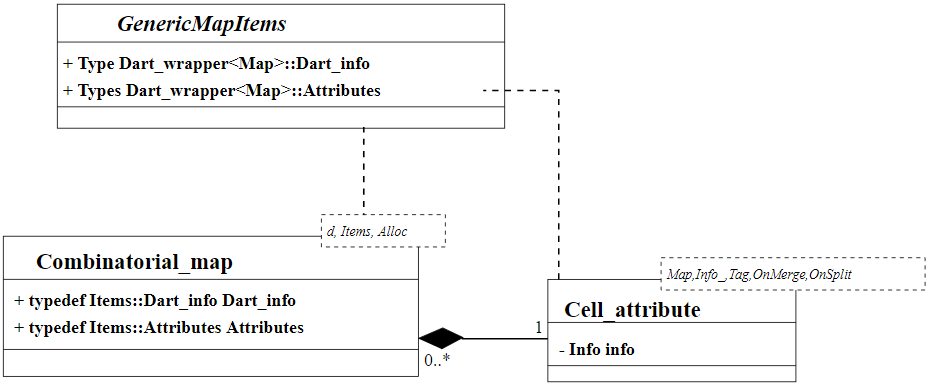
***closed surface***

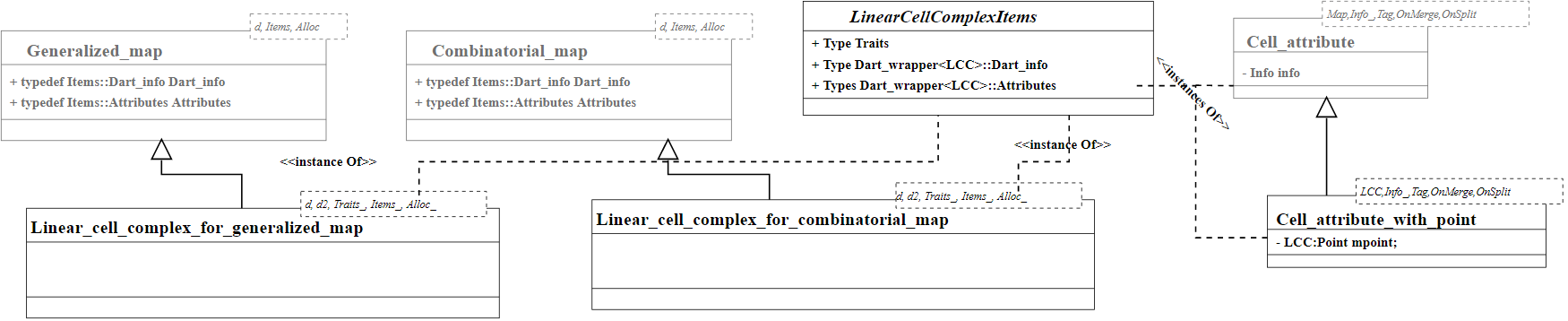
A surface is closed if it contains no border halfedges. A closed surface is a boundary representation for polyhedra in three dimensions.

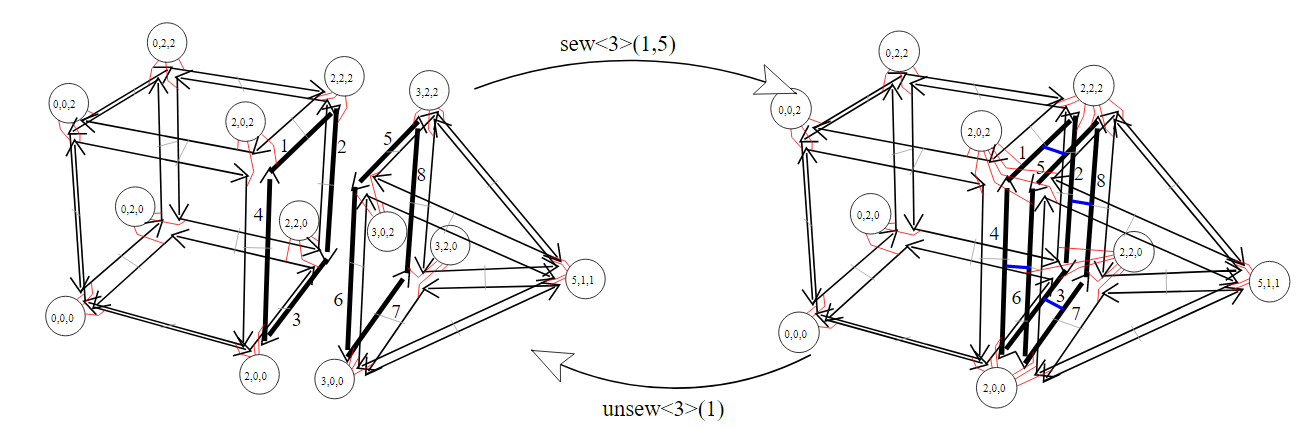
#### Combinatorial Maps and Linear Cell Complex

Damiand, G. and M. Teillaud (2014). "A Generic Implementation of dD Combinatorial Maps in CGAL." Procedia Engineering **82**(Supplement C): 46-58.

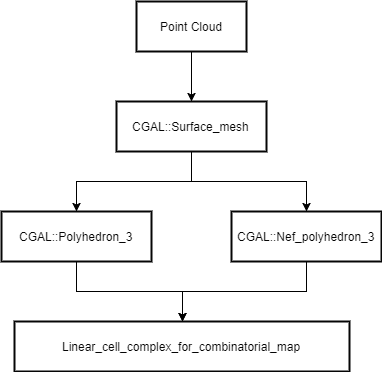
Diakité, A. A., et al. (2014). Topological Reconstruction of Complex 3D Buildings and Automatic Extraction of Levels of Detail. Eurographics Workshop on Urban Data Modelling and Visualisation, Strasbourg, France, Eurographics Association.







#### The processing stage and data structures



# Programming Style Guide

## Naming

### Files

All files should be **under\_scored**.

* Header files have the extension **.h**
* Definition and implementation files have the extension **.hpp**
* Basic classes and functions for efficent consideration
* Templated implementation
* Source files have the extension **.cpp**

### Directories

All directories and subdirectories should be **under\_scored**.

* Header files should go under **include/**
* Templated implementation files should go under **include/impl/**
* Module files should go under **module/**
* **Unit test files should go under utest/**
* **Application files should go under apps/**

### Defines & Macros

Macros should all be **ALL\_CAPITALS\_AND\_UNDERSCORED**.

e.g:

***// the license***

**#ifndef PCL\_MODULE\_NAME\_IMPL\_FILE\_NAME\_HPP\_#define PCL\_MODULE\_NAME\_IMPL\_FILE\_NAME\_HPP\_**

***// the code***

**#endif *// PCL\_MODULE\_NAME\_IMPL\_FILE\_NAME\_HPP\_***

### Namespaces

Namespaces should be **under\_scored**, e.g.:

**namespace ipl{**

**...}**

**namespace ipl {**

**namespace cv {**

**}**

**...}**

**namespace ipl{**

**namespace pc {**

**}**

**...}**

### Classes / Structs

Class names (and other type names) should be **CamelCased**. Exception: if the class name contains a short acronym, the acronym itself should be all capitals. Class and struct names are preferably **nouns**: PFHEstimation instead of EstimatePFH.

Correct examples:

**class ExampleClass;**

**class PFHEstimation;**

### Functions / Methods

Functions and class method names should be **camelCased**, and arguments are **under\_scored**. Function and method names are preferably **verbs**, and the name should make clear what it does: checkForErrors() instead of errorCheck(), dumpDataToFile() instead of dataFile().

Correct usage:

**int applyExample (int example\_arg);**

### Variables

Variable names should be **under\_scored**.

**int my\_variable;**

#### Iterators

Iterator variables should indicate what they’re iterating over, e.g.:

**std::list<int> pid\_list;**

**std::list<int>::iterator pid\_it;**

#### Constants

Constants should be **ALL\_CAPITALS**, e.g.:

**const static int MY\_CONSTANT = 1000;**

#### Member variables

Variables that are members of a class are **under\_scored\_**, with a tailing underscore added, e.g.:

**int example\_int\_;**

### Return statements

Return statements should have their values in parentheses, e.g.:

**int main ()**

**{**

**return (0);**

**}**

## Indentation and Formatting

The standard indentation for each block in PCL is **2 spaces**. Under no circumstances, tabs or other spacing measures should be used. PCL uses a variant of the GNU style formatting.

### Namespaces

In a header file, the contets of a namespace should be indented, e.g.:

**namespace pcl{**

**class Foo**

**{**

**...**

**};}**

In an implementation file, the namespace must be added to each individual method or function definition, e.g.:

**void pcl::Foo::bar (){**

**...}**

### Classes

The template parameters of a class should be declared on a different line, e.g.:

**template <typename T>class Foo{**

**...}**

### Functions / Methods

The return type of each function declaration must be placed on a different line, e.g.:

**void bar ();**

Same for the implementation/definition, e.g.:

**void bar (){**

**...}**

or

**void Foo::bar (){**

**...}**

or

**template <typename T> void Foo<T>::bar (){**

**...}**

### Braces

Braces, both open and close, go on their own lines, e.g.:

**if (a < b){**

**...}else{**

**...}**

Braces can be omitted if the enclosed block is a single-line statement, e.g.:

**if (a < b)**

**x = 2 \* a;**

### Spacing

We’ll say it again: the standard indentation for each block in PCL is **2 spaces**. We also include a space before the bracketed list of arguments to a function/method, e.g.:

**int exampleMethod (int example\_arg);**

If multiple namespaces are declared within header files, always use **2 spaces** to indent them, e.g.:

**namespace foo{**

**namespace bar**

**{**

**void**

**method (int my\_var);**

**}}**

Class and struct members are indented by **2 spaces**. Access qualifiers (public, private and protected) are put at the indentation level of the class body and members affected by these qualifiers are indented by one more level, i.e. 2 spaces. E.g.:

**namespace foo{**

**class Bar**

**{**

**int i;**

**public:**

**int j;**

**protected:**

**void**

**baz ();**

**}}**

### Automatic code formatting

The following set of rules can be automatically used by various different IDEs, editors, etc.

#### Emacs

You can use the following [PCL C/C++ style file](https://raw.githubusercontent.com/PointCloudLibrary/pcl/master/doc/advanced/content/files/pcl-c-style.el), download it to some known location and then:

* open .emacs
* add the following before any C/C++ custom hooks

**(load-file "/location/to/pcl-c-style.el")(add-hook 'c-mode-common-hook 'pcl-set-c-style)**

#### Uncrustify

You can find a semi-finished config for [Uncrustify](http://uncrustify.sourceforge.net/) [here](http://dev.pointclouds.org/attachments/download/537/uncrustify.cfg)

#### Eclipse

You can find a PCL code style file for Eclipse [on GitHub](https://github.com/PointCloudLibrary/pcl/tree/master/doc/advanced/content/files).

To add the new formatting style go to: Windows > Preferences > C/C++ > Code Style > Formatter

To format portion of codes, select the code and press Ctrl + Shift + F.

If you want to format the whole code in your project go to the tree and right click on the project: Source > Format.

Note that the Eclipse formatter style is configured to wrap all arguments in a function, feel free to re-arange the arguments if you feel the need; for example, this improves readability:

**int displayPoint (float x, float y, float z,**

**float r, float g, float b**

**);**

This eclipse formatter fails to add a space before brackets when using PCL macros:

**PCL\_ERROR("Text\n");**

should be

**PCL\_ERROR ("Text\n");**

**Note**

This style sheet is not perfect, please mention errors on the user mailing list and feel free to patch!

## Structuring

### Classes and API

For most classes in PCL, it is preferred that the interface (all public members) does not contain variables and only two types of methods:

* The first method type is the get/set type that allows to manipulate the parameters and input data used by the class.
* The second type of methods is actually performing the class functionality and produces output, e.g. compute, filter, segment.

### Passing arguments

For get/set type methods the following rules apply:

* If large amounts of data needs to be set (usually the case with input data in PCL) it is preferred to pass a boost shared pointer instead of the actual data.
* Getters always need to pass exactly the same types as their repsective setters and vice versa.
* For getters, if only one argument needs to be passed this will be done via the return keyword. If two or more arguments need to be passed they will all be passed by reference instead.

For the compute, filter, segment, etc. type methods the following rules apply:

* The output arguments are preferably non-pointer type, regardless of data size.
* The output arguments will always be passed by reference.

# Applications

注意：点云预处理时，需要对坐标进行中心化，避免精度损失