# Introduction

The “image.h” is the core of the PCL library, which defines how an image is stored.

A unique behavior of images in PCL library is that it allows for an arbitrary starting coordinate, in contrast to the common fixed starting coordinate of (0, 0, 0).

A typical image object is defined in three layers, which are:

1. ImageBase
2. ImagePhysicalLayer
3. Image

## ImageBase

This is the basis of all images in PCL. It contains general information on the voxel layout of the image, as in size and region, and conversion between an index and a point.

This is also a non-template class, therefore it can be used as a container to store all types of images in an OO fashion.

This class also have a generic getValue() and setValue() method that enable getting and setting of voxel values, though on is limited to only double (A not a number will be returned if the underlying values cannot be converted to double).

**Note**: Due to the inherent slowness of getValue() and setValue(), these two methods should only be used as a last resort.

## ImagePhysicalLayer

This layer contains the physical description of the image, which consists of origin, spacing and orientation.

The convenience method for converting between image coordinates and physical coordinates is also defined in this layer.

**Note**: The ImagePhysicalLayer class contains a Boolean template argument, which is used for enabling and disabling the usage of orientation. If orientation is not of concern, there is a need of conversion between image and physical coordinates and speed is a concern, then the usage of orientation should be disabled. Else, orientation should be enabled.

## Image

The actual image object, which contains most of the actual implementation, which contains:

* get() and set() method for efficient access to voxel values, which are the preferred way for accessing values in the image.
* localToIndex() method which is more efficient than the toIndex() method in ImageBase.
* getAlias() method
  + All attributes related to the physical interpretation of an image (minimum point, origin, spacing and orientation) is held constant for an image
  + The getAlias() method provides the modification of physical interpretation related attributes by returning an alias image with the modified attributes, in which the new and original image shares the same memory space for the voxel data
* getSubImage() method for creating an alias image that covers a smaller region than the original image.
* getWholeImage() method for getting the whole image if the current image is a result of getSubImage().

Typically, this is the pcl::Image class in <pcl/image.h>, but it can also be some other classes (e.g. DummyImage).

# Creating an image from scratch

Creation of image is done via the New() static method. Below is an example for creating a 10x10x10 image of “char” values with the orientation enabled:

#include <pcl/image.h>

auto image = pcl::Image<char,true>::New(pcl::Point3D<int>(10,10,10));

The created image will start from (0,0,0) and end at (9,9,9). Alternatively, if a different starting point is preferred, a minimum and maximum point can be used instead:

auto image = pcl::Image<char,true>::New(pcl::Point3D<int>(5,5,5), pcl::Point3D<int>(15,15,15));

where the created image will span from (5,5,5) to (15,15,15).

Physical interpretation related attributes can also be specified via New() as follows:

auto image = pcl::Image<char,true>::New(

pcl::Point3D<int>(5,5,5), pcl::Point3D<int>(15,15,15),

pcl::Point3D<double>(10.5, 15.5, 20.5), // This is the origin

pcl::Point3D<double>(0.5, 0.5, 0.5), // This is the spacing

orientation\_matix // A vnl\_matrix\_fixed<double,3,3>

);

When not specified, origin is defaulted to (0,0,0), spacing is defaulted to (1,1,1) and orientation is an identify matrix.

A new image can also be created based on an existing image, for example:

auto new\_image = pcl::Image<double,true>::New(image)

Except for the voxel value type and the actual voxel values (which is not set by default), the size and physical representation of “new\_image” will be identical to “image”.

# Accessing voxel values

Below is how to set and get the value of a voxel at position (10,10,10):

image->set(pcl::Point3D<int>(10,10,10), 100);

std::cout << image->get(pcl::Point3D<int>(10,10,10)); # 100

A faster method however is specify via an index, for example:

Image->set(0L, 100);

Where 0 represents the first voxel. The last voxel is sizeX × sizeY × sizeZ.

To iterate through an image, it is recommended to use the ImageIterator from object:

ImageIterator iter(image);

pcl\_ForIterator(iter) {

image->set(iter, 10);

}

# Coordinates conversion

Below are examples on how one can convert between image coordinates, index and physical coordinates:

# Slower version as this is defined in ImageBase

long index = image->toIndex(pcl::Point3D<int>(10,10,10));

std::cout << toPoint(index); // (10,10,10)

# Faster version directly defined in Image

index = image->localToIndex(pcl::Point3D<int>(10,10,10));

std::cout << localToPoint(index); // (10,10,10)

auto physical\_point = image->toPhysicalCoordinate(pcl::Point3D<int>(10,10,10));

std::cout << image->toImageCoordinate(physical\_point); // (10,10,10)

# Alias

Below is an example on how one can change the minimum point of an image to (15,12,13):

auto new\_image = image->getAlias(pcl::Point3D<int>(15,12,13), true);

The extra Boolean value of “true” next to the new minimum point indicates that the physical representation are to be preserved (a new origin will be assign to preserve). If the a “false” is provided, the original origin will be used, effectively translating the image in the physical coordinates.

Below is an example where the spacing of “new\_image” is changed to (1.5,1.5,2), while everything remains unchanged:

auto new\_image = image->getAlias(image->getMinPoint(), pcl::Point3D<double>(1.5,1.5,2));

# Sub image

Below is an example of getting a sub image:

auto sub\_image = image->getSubImage(pcl::Point3D<int>(3,3,3), pcl::Point3D<int>(5,5,5));

The provided minimum and maximum point will be automatically clamp to the ones in “image” if the limit is exceeded.

To recover the whole image from a sub image, simple do the following:

auto whole\_image = sub\_image->getWholeImage();

# ImageHelper

ImageHelper is a class that contains various convenience methods in the form of static functions. For example, to fill a newly created image with zeros, simple do the following:

pcl::ImageHelper::Fill(image, 0);

Below is an example obtain a cropped version of an image, where a value of 20 will be assigned if region outside the image is specified:

auto new\_image = pcl::ImageHelper::GetCroppedAuto(

image,

pcl::Point3D<int>(10,10,10),

pcl::Point3D<int>(100,100,100),

20

);

It should be noted that unlike getSubImage(), the newly created “new\_image” is a separated entity from “image”.

It is also possible to specify the voxel type of the cropped version that may be different from the original image:

auto new\_image = pcl::ImageHelper::GetCropped<pcl::Image<int,true>>(

image,

pcl::Point3D<int>(10,10,10),

pcl::Point3D<int>(100,100,100),

20

);

Below is an example to get a copy of an image:

auto image\_copy = pcl::ImageHelper::GetCopyAuto(image);

auto copy\_with\_double = pcl::ImageHelper::GetCopy<pcl::Image<double,true>>(image);

## Conversion between PCL and ITK

ImageHelper contains functions for conversion between PCL Image to ItkImage.

Below is how one can get an ITK alias and copy from Image:

auto itk\_alias = pcl::ImageHelper::GetItkImage(image, false); //Ownership is not transferred

auto itk\_copy = pcl::ImageHelper::GetItkImageCopy(image);

Below is how to convert from ItkImage to PCL Image:

auto image = pcl::ImageHelper::CreateFromItkImage<pcl::Image<char,true>>(

itk\_image,

true //Ownership is transferred to image if an alias is possible

);

The “image” will be an alias of “itk\_image” only if the voxel values of “itk\_image” is of type “char”. Else a copy will be provided. To create a copy, the CreateCopyOfItkImage() can be used instead.