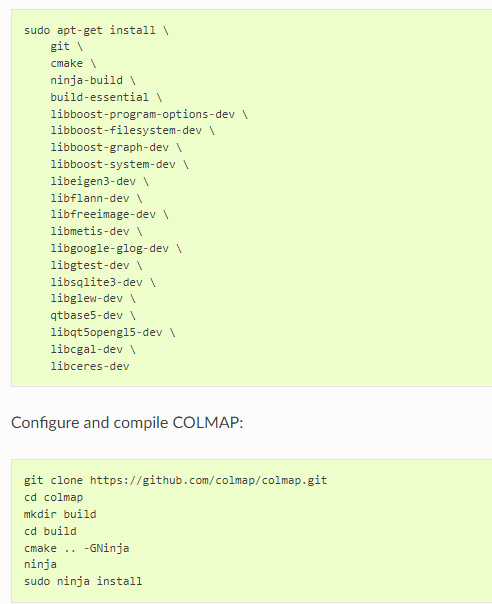
# Colmap SFM部分

官方数据手册：https://colmap.github.io/

## 官方文档关键信息

### 1.0 从源码编译



**需要用户自行安装的三方库：**

Build-essential 包括了：g++编译器for C++、gcc编译器for C、make编译命令、libc6-dev用于编译C++和C的头文件。

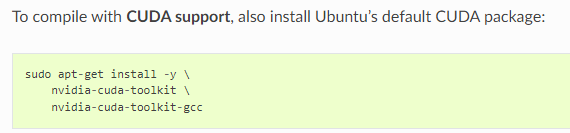
Freeimage：是一个开源工具，封装了对各种图像的操作。

Metis：是一个开源的图割开源算法，<https://github.com/KarypisLab/METIS/>；http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

Sqlite3：是一个开源数据，https://github.com/sqlite/sqlite

Glew：是一个对OpenGL封装的开源库，<https://glew.sourceforge.net/>

如果需要依赖cuda则还要安装下面两个库：



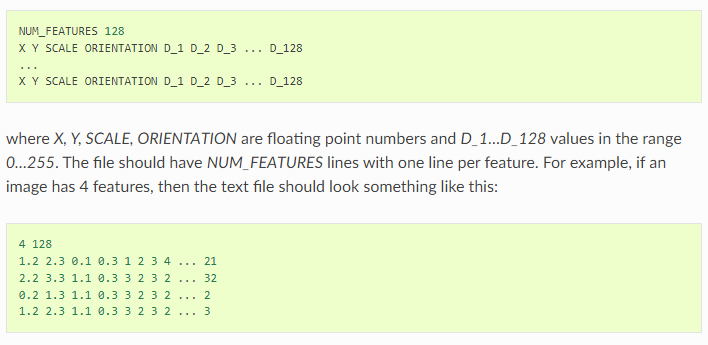
### 1.1 database management

Keypoint会存储两个点的坐标和feature geometry信息。feature geometry存储的信息分成两种情况：第一种情况存储的是尺度和特征点的方向；第二种情况存储的是affinity shape。

描述子存储的是128维度的向量，每个元素都是uint8。

如果图像中包含EXIF 信息，这个信息中通常会包括拍摄相机的内参信息，这时colmap可能会直接使用EXIF中的相机内参数值。

每个图像会提取特征点，每张图像的特征点信息会保存成一个txt文件：



如果你让colmap提取所有图像的特征点，还没有完成时就取消了，那么下次这个工程再加载时会从上次没有提取的图像开始进行提取。

### 1.2 Colmap中匹配模式选择

Exhaustive Matching：所有图像会进行匹配；

Sequential Matching：如果输入的图像是有序的，那么图像的名称必须按照序号进行排列，回环会进行触发。但是你需要先从<https://demuc.de/colmap/#tutorial>下载预训练的词袋模型；

Vocabulary Tree Matching：先用词袋模型计算所有图像的相似度，然后再针对相似的图像进行匹配，这种方式建议在大尺度的重建场景中使用；

Spatial Matching: 如果图像中的EXIF 信息包括GPS信息，那么会根据图像的GPS位置来进行图像的匹配；

Custom Matching:用户自定义匹配对定义，用户指定哪两个图像需要进行匹配；

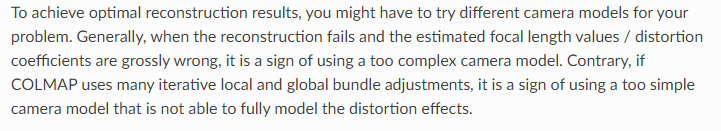
### 1.3 Colmap中的相机模型

SIMPLE\_PINHOLE, PINHOLE：输入的图像已经畸变矫正过了；

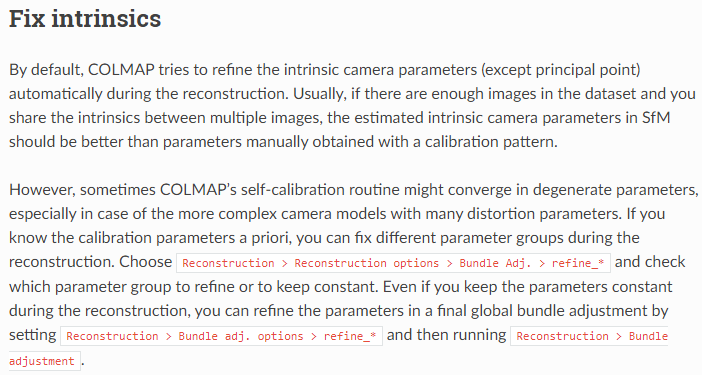
SIMPLE\_RADIAL, RADIAL：相机内参未知，并且每个图像的内参可能都会不同；

OPENCV, FULL\_OPENCV：如果提前知道标参数，并作为初值，用于后续的优化；

SIMPLE\_RADIAL\_FISHEYE, RADIAL\_FISHEYE, OPENCV\_FISHEYE, FOV, THIN\_PRISM\_FISHEYE：用于鱼眼模型；



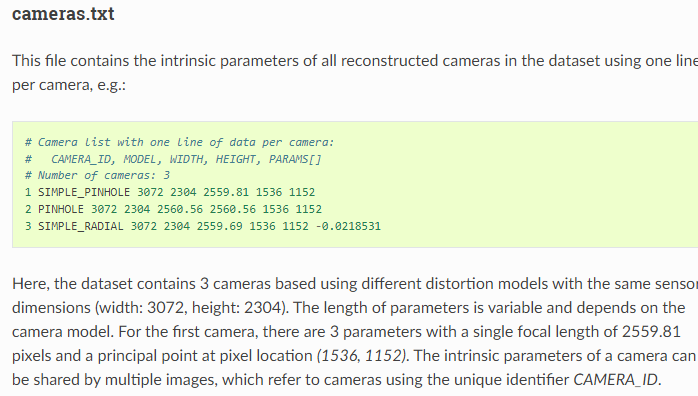
优化的过程中固定相机内参设置：

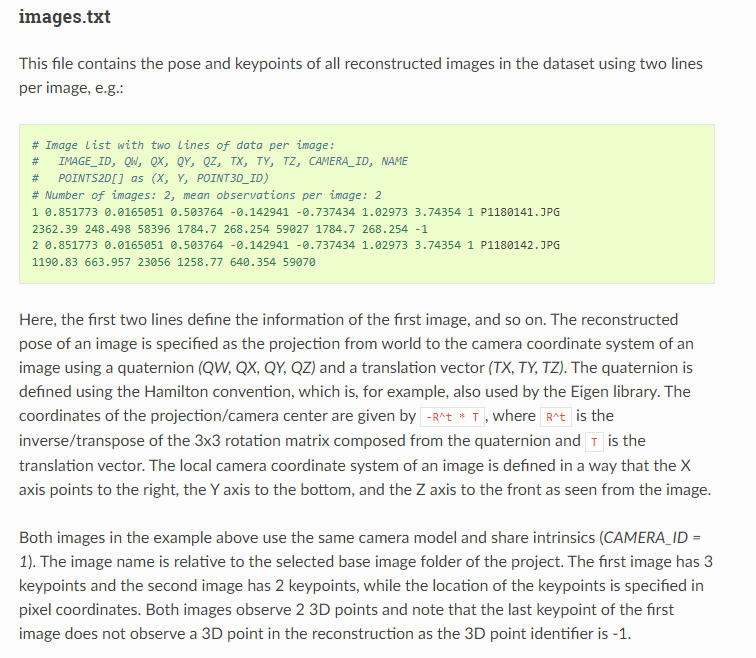


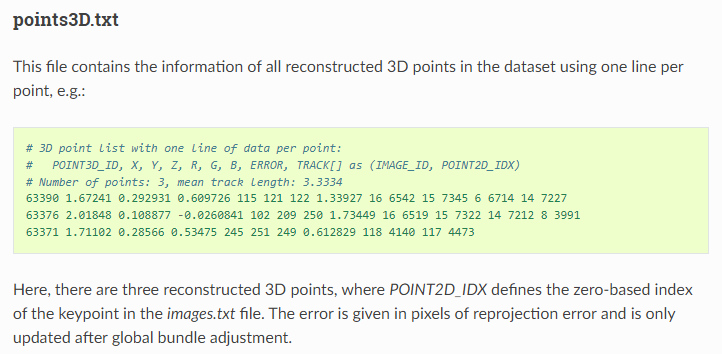
### 1.4 数据导出

Colmap能够导出各种数据，导出的数据结构详见：<https://colmap.github.io/format.html#output-format>. 导出的格式可以是二进制或者是方便人阅读的txt。

Colmap导出的数据分为三个部分：相机，特征点和图像：







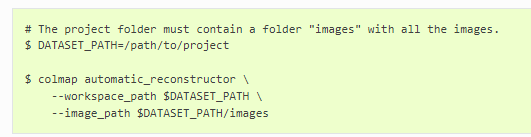
### 1.5 数据集下载

<https://demuc.de/colmap/datasets/>

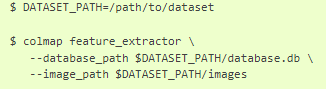
### 1.6 启动命令

<https://colmap.github.io/cli.html>

不打开可视化工具进行自动稀疏和稠密重建：



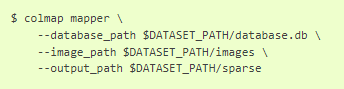
特征提取：



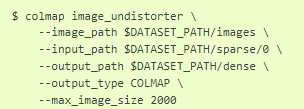
匹配：



Sfm稀疏重建：



图像畸变矫正：



## FeatureKeypoint仿射矩阵

对于一个特征点可以使用xy方向的尺度变量(scale\_x, scale\_y)、旋转矩阵(theta)和剪切变量(shear)，完整的2\*2的仿射矩阵定义如下：

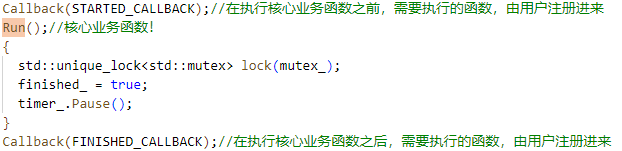
## TwoViewGeometry数据结构

需要使用一个数据结构来存储两个图像之间的几何位姿变换，其中包括：essential matrix, fundamental matrix，se3位姿变换和单应性矩阵。

## 任务队列实现分析

在Colmap中作者为了简化任务流，实现了一个抽象的队列和各个任务线程。作者实现了一个Thread类，在这个类中有两个核心的函数和一个变量：

**Start函数**：是一个框架函数，会先后调用**callbacks\_**变量中注册的函数和**Run**函数，这个函数一旦调用表示会启动一个线程执行如下图所示的代码：

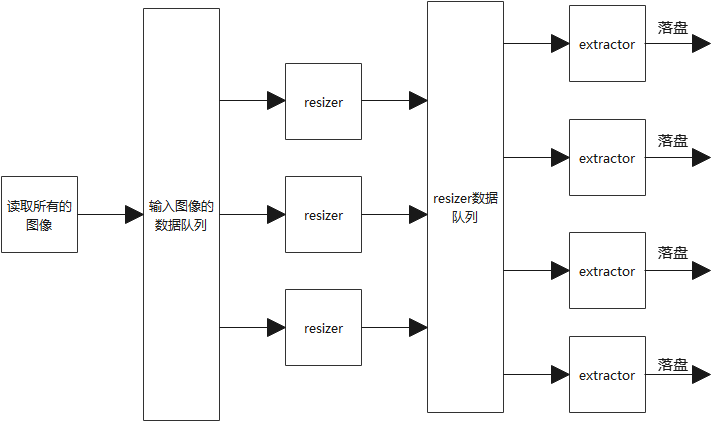


**Run函数**：当子类继承Thread后，只要覆写这个函数即可，本质上这个函数里面的内容是实际执行的代码。这个函数是核心业务逻辑函数。

**callbacks\_变量**：在RunFunc函数执行之前和之后，可能用户都想要执行一些特别的任务，Thread类提供了这样的一个函数变量，允许用户将想要执行的函数注册进来。

这么做的函数是将算法的核心代码和业务的数据流转彻底分离开。

但是如何实现数据的流转呢？例如特征点提取任务由resize图像和在图像上提取特征点两个部分组成。在colmap实现的过程中，会启动N个resize图像线程，和M个图像上提取特征点线程，下图所示我们举例N=3，M=4。



我们想要达成的效果是将图像一张张读取进来，然后塞入到输入图像队列中，然后resizer从此队列中取数据并进行处理，处理完成后将数据放入到resizer队列中，然后extractor从此数据队列取数据，extractor完成任务后将处理完成的数据进行落盘。

那么colmap中是如何实现上面的功能的呢？

1.在FeatureExtractorController类的构造函数先构建数据的流向：



上图中红色方框圈出来的数据，就是作者自己定义的用于多线程进行共享的数据队列。

2.然后在启动函数中，启动N个resizer线程和M个特征提取线程。

3.最后从硬盘中读取图像，塞入到输入图像数据队列，对应上面代码的resizer\_queue变量。

## 特征初始匹配

## 特征匹配几何校验

## DetectWatermark