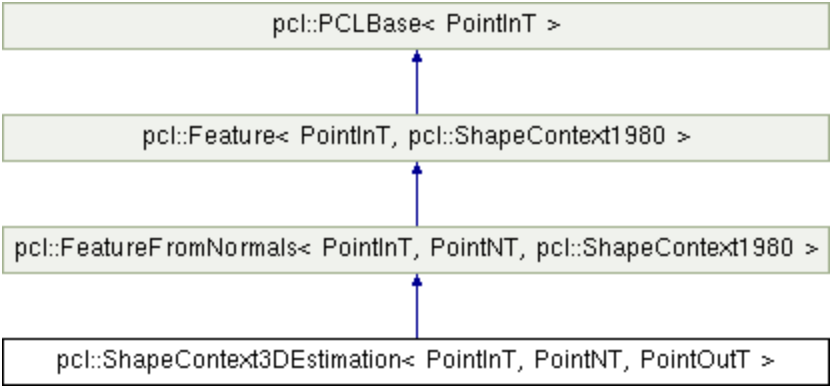
**class pcl::ShapeContext3DEstimation<PointInT, PointNT, PointOutT>**

一、类的功能

该类实现了3D形状内容描述子的相关算法。

若不能计算某点的最近邻居，或某点的x,y或z轴的坐标是NaN，则将该点的描述子的值置为NaN，且将该点的参照系的值置为0。

二、类的继承关系



三、类的关键函数接口

1.进行一些初始化的计算

bool pcl::ShapeContext3DEstimation<PointInT, PointNT, PointOutT>::initCompute ()

2.计算给定点的描述子

bool pcl::ShapeContext3DEstimation<PointInT, PointNT, PointOutT>::computePoint (

    std::size\_t index, const pcl::PointCloud<PointNT> &normals, float rf[9], std::vector<float> &desc)

3.计算点云的特征

void pcl::ShapeContext3DEstimation<PointInT, PointNT, PointOutT>::computeFeature (PointCloudOut &output)

4.设置沿方向角划分的数目

Void setAzimuthBins(size\_t bins)

5.设置沿俯仰角划分的数目

Void setElevationBins(size\_t bins)

6.设置沿径向划分的数目

Void setRadiusBins(size\_t bins)

7.设置最小径向半径

Void setMinimalRadius(double radius)

8.设置输入点云对应的法线数据对象

Void setInputNormals(const PointCloudNConstPtr &normals)

9.计算输出

Void compute(PointCloudOut &output)

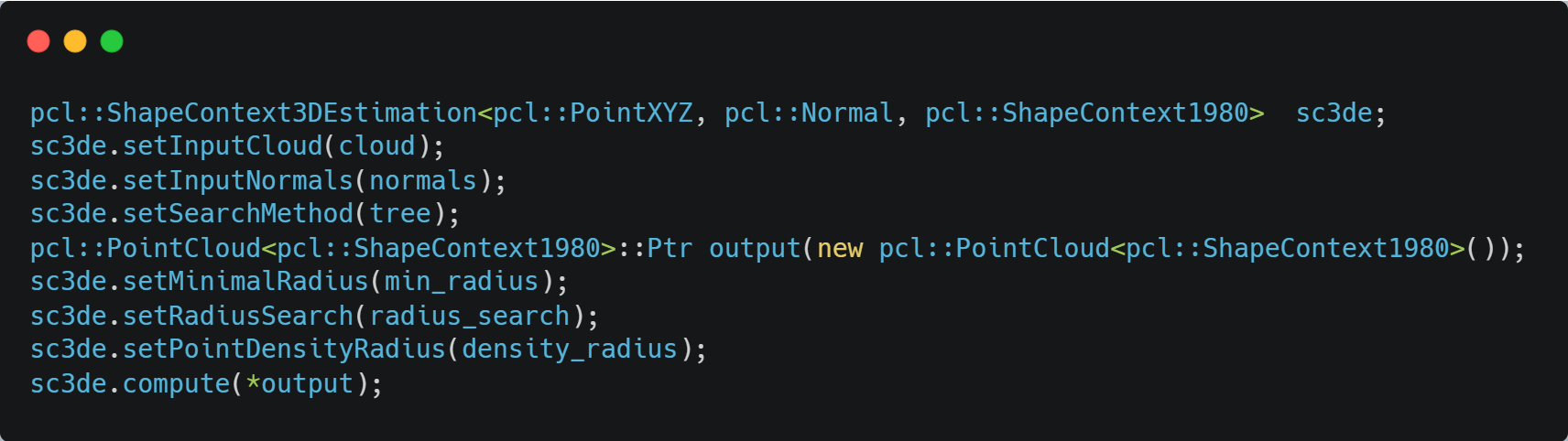
1. 函数接口分析的详细分析

1.实际应用中接口的调用方式

1.1说明：

ShapeContext3DEstimation中实现了3个比较重要的函数，分别是initCompute()、computePoint()以及computeFeature()，由于在computeFeature()的内部调用到了computePoint()，因此先对computeFeature()进行分析。我们先从实际的开发场景入手。

图 1 实际开发中接口的调用方式



1.2解释说明图1：

1. 声明ShapeContext3DEstimation的实例sc3de（第1行）
2. 设置一系列参数（第2~4行）
3. 声明输出点云的实例output（第5行）
4. 设置一系列参数（第6~8行）
5. 计算点云的特征（第9行）

1.3解释说明sc3de.compute(\*output)（第9行）：

1. 由类的继承关系得知，ShapeContext3DEstimation基类的基类为Feature，而compute(PointCloudOut &output)正是由Feature实现的函数。
2. 下面对Feature<PointInT, PointOutT>::compute (PointCloudOut &output)函数内部的调用逻辑进行说明：

表 1 compute函数内部的调用顺序

1.Feature<PointInT, PointOutT>::initCompute ()

2.pcl::ShapeContext3DEstimation<PointInT, PointNT, PointOutT>::computeFeature (PointCloudOut &output)

2.1pcl::ShapeContext3DEstimation<PointInT, PointNT, PointOutT>::computePoint (

    std::size\_t index, const pcl::PointCloud<PointNT> &normals, float rf[9], std::vector<float> &desc)

3.Feature<PointInT, PointOutT>::deinitCompute ()

1.4解释说明表1：

（1）首先调用initCompute，功能为校验一系列参数

（2）然后调用computeFeature，计算点云的特征。computeFeature函数中嵌套了 computePoint函数，作用是计算某点的3D形状内容描述子。

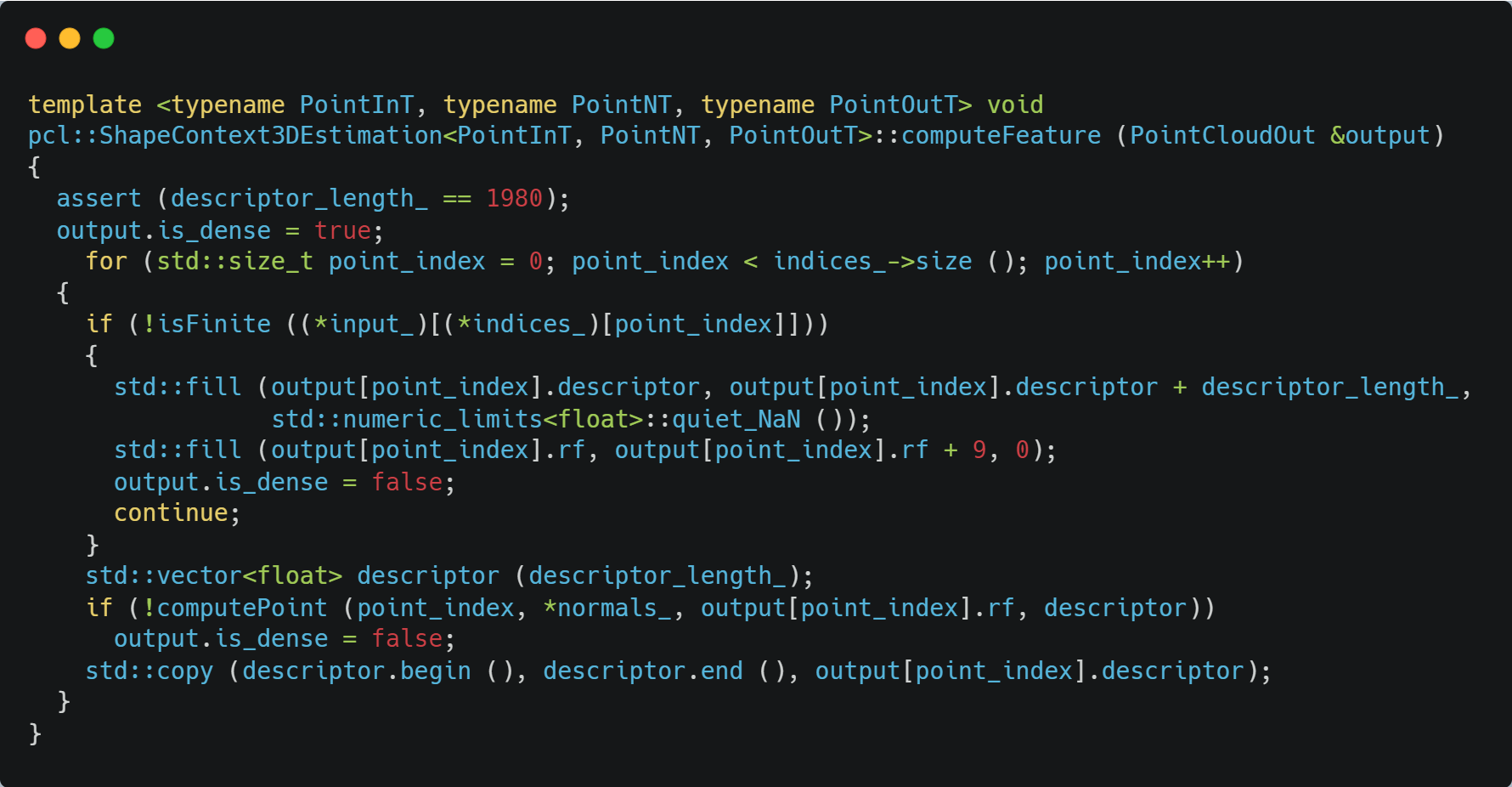
（3）最后调用deinitCompute，作用尚不明确

2.computeFeature()接口分析

2.1接口功能描述

计算点云的特征

2.2接口代码



2.3接口逻辑描述（伪代码）

1. 校验点类型(ShapeContext1980)
2. 对所有需要计算的点执行以下判断（循环）
   1. 对于无效点
      1. 将描述子的值设为NaN
      2. 将参照系的值置为0
      3. 探索下一个点
   2. 否则，对于有效点
      1. 声明一个描述子，记d
      2. 计算该点的描述子，将结果赋给d
      3. 用d更新输出点云的描述子
      4. 探索下一个点
3. 循环执行完毕，计算结果更新到output

2.4接口参数说明

1. PointInT: 输入点云的点的类型
2. PointNt: 点的法线类型
3. PointOutT: 输出点云的点的类型
4. output:
   1. 描述:算子入参
   2. 类型:PointCloud
   3. 属性:
      1. is\_dense(bool): true if no points are invalid.
      2. points(vector):表示点云中的点, PointCloud类重载了运算符[]。在该接口中 points要求的类型是ShapeContext1980.
5. input\_:
   1. 描述:一个指向输入点云对象的一个指针
   2. 类型:shared\_ptr<const PointCloud<PointT> >
6. indices\_:
   1. 描述:一个指向由要使用的点组成的vector的指针
   2. 类型: shared\_ptr<Indices>
7. shared\_ptr:
   1. 描述:c++11的智能指针，可自动释放分配的内存
8. 若要用Tik实现，则必须解决的问题：

1.接口入参output为c++对象，如何将c++对象及其属性映射到tik中。

（1）思路：将c++的对象扁平化处理，即将对象的属性的及其值映射成字典数据。

（2）举例：computeFeature(\*output)的入参output的类型为PointCloud，PointCloud具 有is\_dense,points等属性。将is\_dense映射为{‘is\_dense’: bool}，将points映射为 {‘points’:list(float)}。

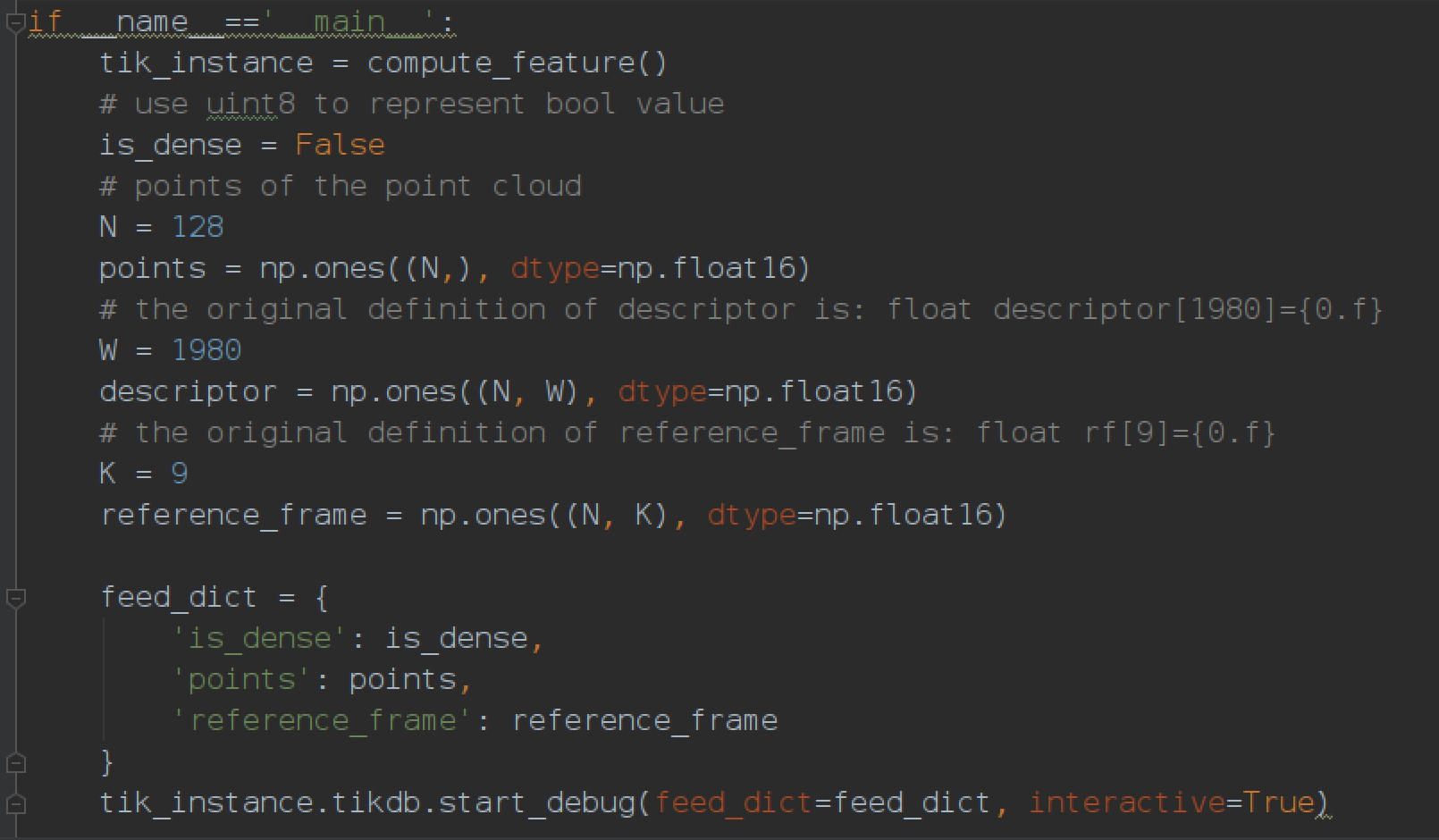
2.若入参的属性也是一个结构体，即入参是一个嵌套的结构，如何处理？例如points具 有descriptors和rf这两个属性

（1）思路：将points映射为多维dict，对应多维Tensor

（2）举例：假设有N个points，每个point中各有一个descriptor和一个rf，descriptor 是一个长度为W的向量，rf是一个长度为K的列表。

①可以将N个points映射为Tensor(, (N, W+K),)

②将points的属性拆开，分别定义成一个2维Tensor，如下图所示



1. 接口里调用了许多其他的函数（如isFinite()），如何对此类函数进行封装？

pcl::ShapeContext3DEstimation<PointInT, PointNT, PointOut>::computePoint (std::size\_t index, const pcl::PointCloud<PointNT> &normals, float rf[9], std::vector<float> &desc)

实际上用到的变量

std::size\_t index

pcl::PointCloud<PointNT> normals

float rf[9]

std::vector<float> desc

PCLBase<PointNT> input\_

PCLBase<PointNT> indices\_

PointCloudInConstPtr surface\_

std::size\_t radius\_bins\_

double search\_radius\_

std::size\_t elevation\_bins\_

|  |  |  |
| --- | --- | --- |
| 行号 | 解释 | Tik接口 |
| 6~7 | 用指向float数组的指针初始化Map |  |
| 11~12 | 定义两个变量（nn\_indices用于存储临近点的下标；nn\_dists用于存储临近点的距离） |  |
| 13 | 调用searchForNeighbors计算临近点的数量（找不到这个函数的源码？），共有3个返回值（neighb\_cnt返回临近点的数量；nn\_indices储存所有临近点的下标；nn\_dists储存所有查询点到临近点的距离） | for\_range |
| 14~19 | 接口的第一个出口。当neighb\_cnt==0时，置descriptor=NaN，rf=0.f，return false |  |
| 21~22 | 得到距查询点最近的点的下标 | for\_range, scalar\_min |
| 25 | 获取查询点（即同心球的球心） | vec\_sub |
| 28~33 | 接口的第二个出口。当查询点的法线不有限时，置descriptor=NaN，rf=0.f，return false |  |
| 34 | 获取查询点的法线 |  |
| 37~39 | 对x轴的全部三个元素用随机数初始化 |  |
| 40~45 | 计算x轴的坐标（查询点(0,0,0)应该作为向量的头）（猜测）  假设x轴与法线正交，X1X2+Y1Y2+Z1Z2=0反推出x轴的坐标(X1,Y1,Z1)  if：法线的z坐标不为0.f  then：Z1=-(X1X2+Y1Y2)/Z2  if：法线的y坐标不为0.f  then：Y1=-(X1X2+Z1Z2)/Y2  if：法线的x坐标不为0.f  then：X1=-(Y1Y2+Z1Z2)/X2 | vec\_mul,  vec\_add,  vec\_rec,  vec\_sub |
| 47 | 对x轴做向量标准化 | vec\_rec |
| 50 | 检查经过计算后的x轴是否与查询点的法线正交 | vec\_mul,  vec\_add,  vec\_sub |
| 53 | 计算y轴：y轴=法向量与x轴做向量积（cross product），由于x轴与法线正交，他们两个再做×积得到y轴，则x、y与法线皆两两正交。 | vec\_mul,  vec\_sub,  vec\_add |
| 56~115 | 遍历所有临近点 | for\_range |
| 58~59 | 首先排除距离为0的临近点 |  |
| 61 | 获取当前临近点的坐标 |  |
| 65 | 获取查询点与当前临近点的距离r（这里用了平方根，尚不清楚具体意义） | scalar\_sqrt |
| 68 | 定义一个变量proj，表示临近点在法平面的投影 |  |
| 69~70 | 计算临近点在法平面的投影proj | vec\_sub,  vec\_mul,  vec\_add |
| 73 | 对投影向量proj做标准化处理 | vec\_rec |
| 76~78 | 计算投影proj与x轴的夹角phi∈[0,360]  pcl::rad2deg：将弧度转换成角度  std::atan2：返回y/x的反正切值，返回值以弧度表示 | vec\_mul,  vec\_sub,  vec\_add,  tik中似乎没有反三角函数和三角函数 |
| 80~83 | 计算临近点与法线（z轴）的夹角theta∈[0,180]  std::acos：反余弦函数，返回值以弧度表示 | vec\_mul,  vec\_sub,  vec\_add,  tik中似乎没有反三角函数和三角函数 |
| 86 | 从所有半径区间(radii\_interval\_)中找出第一个不小于r（第65行定义）的元素，返回值是指向该元素的iterator，记作rad\_min | for\_range, scalar\_min |
| 87 | 从所有theta区间(theta区间的划分与俯仰角elevation相关)中找出第一个不小于theta(在第83行定义)的元素，返回值是指向该元素的iterator，记作theta\_min | for\_range, scalar\_min |
| 88 | 从左右phi区间(phi区间的划分与转动叫azimuth相关)中找出第一个不小于phi(phi表示一个实际的转动角的值，在第77~78行定义并计算出)的元素，返回值是指向该元素的iterator，记作phi\_min | for\_range, scalar\_min |
| 91~93 | 计算rad\_min,theta\_min,phi\_min所在区间的下标 | vec\_sub |
| 96 | 定义一个储存当前临近点的临近点的下标的容器neighbour\_indices |  |
| 97 | 定义一个储存当前临近点距其他临近点的距离的容器neighbour\_distances |  |
| 98 | 调用searchForNeighbour方法，由3个输入、3个输出。3个输入分别是（\*surface：一个静态点云对象指针，尚不清楚具体含义；nn\_indices[ne]：当前临近点的下标；point\_density\_radius：当前临近点搜索其临近点的搜索半径），3个输出分别是（neighbour\_indices：存储当前临近点搜索到的临近点的下标；neighbour\_distances：存储当前临近点距搜索到的临近点的距离； point\_density：返回当前临近点搜索到的临近点的数量） | for\_range |
| 100 | 若当前临近点查询不到point\_density\_radius范围内的其他点，则直接查询下一个临近点。 |  |
| 103 | 计算w，尚不知道具体意义 | vec\_rec,  vec\_mul,  vec\_add |
| 106 | 检查w是否大于等于0 |  |
| 107~110 | 检查w是否合法（不合法的情况：w==infinity；w==NaN） |  |
| 112 | 更新查询点的描述子：descriptor[l][k][j] += w | vec\_mul,  vec\_add |
| 114 | 检查更新后的查询点的描述子descriptor是否大于等于0 |  |
| 118 | 将rf置0，尚不清楚具体意义 |  |
| 119 | 接口运行正确时的出口。return true |  |