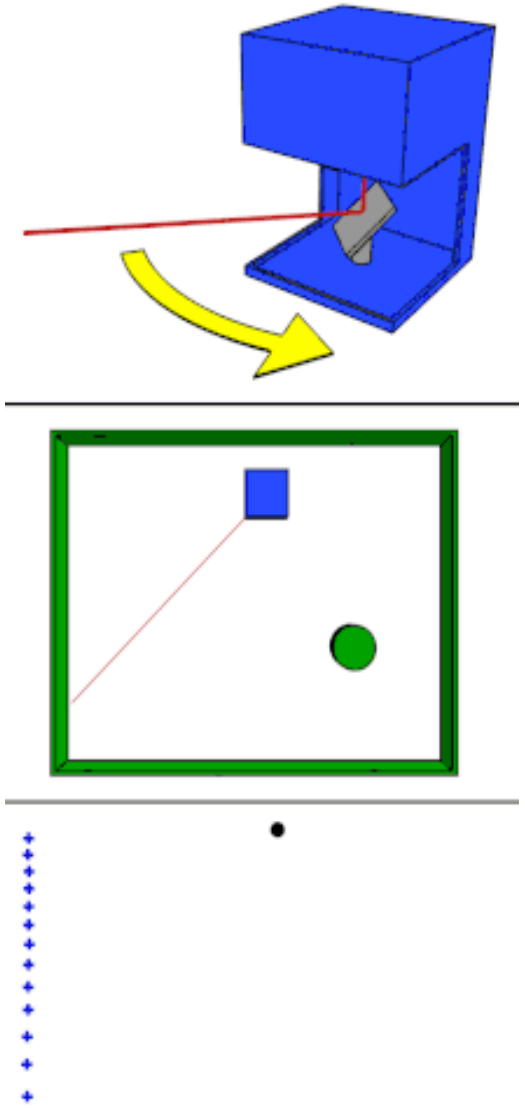
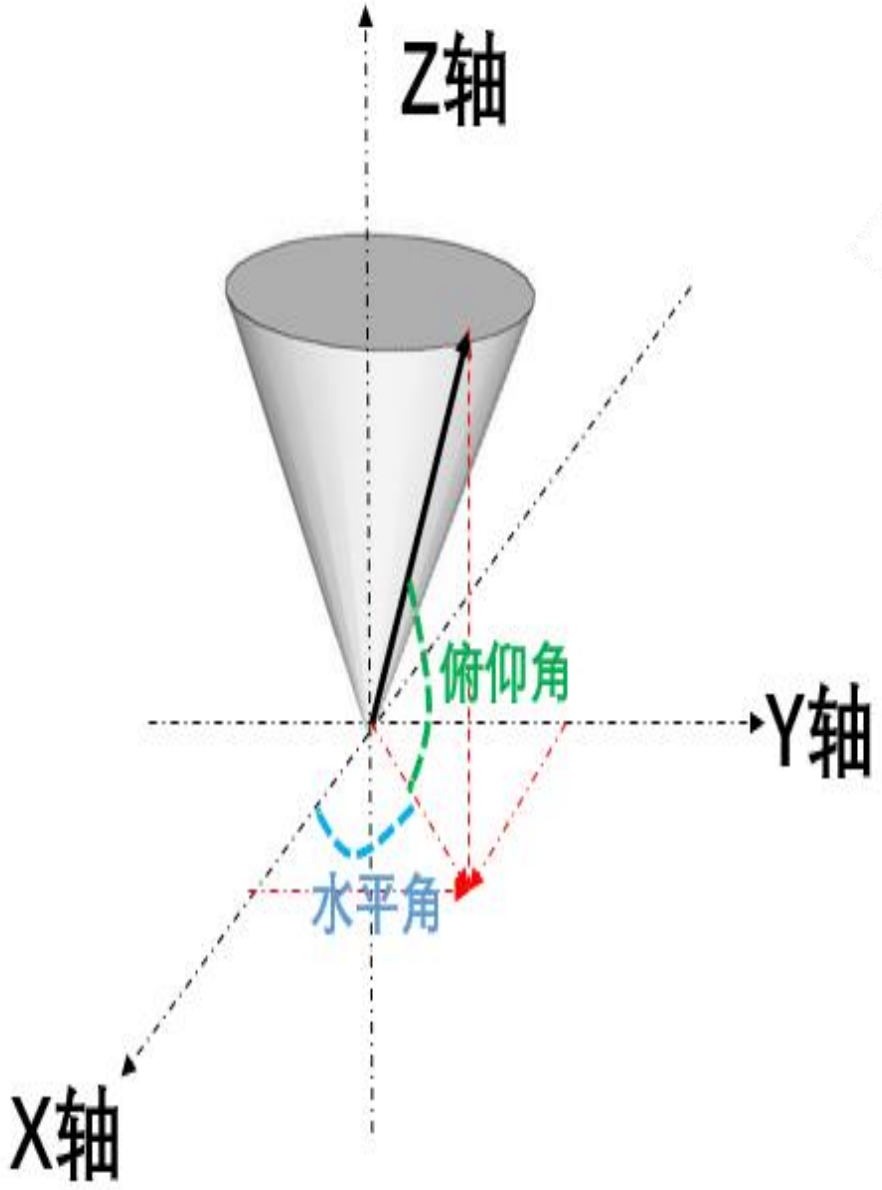
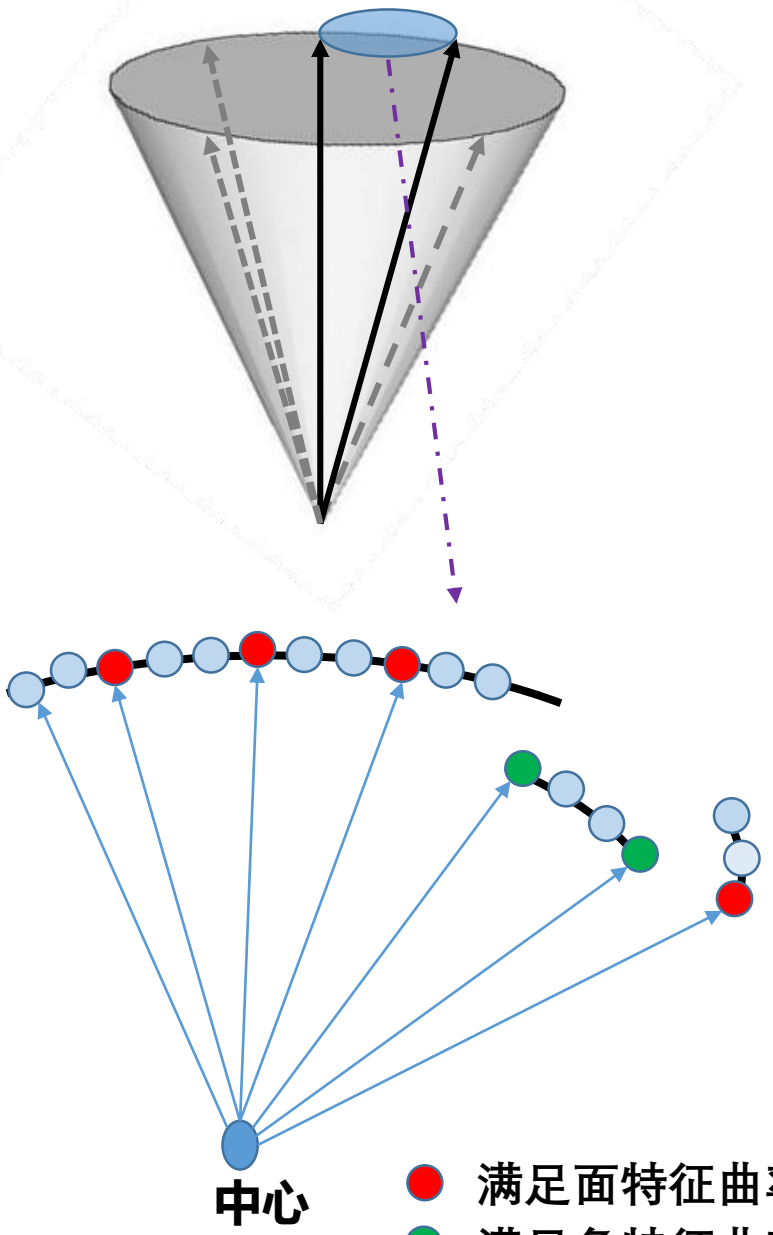
Lidar_feature

雷达下feature的提取







i是点在点云当中的序号

第i个点曲率：cloudCurvature[i]
第i个点状态：cloudNeighborPicked[i];
第i个点属性：cloudLabel[i];
第i个点关联容器：
cloudSmoothness[i].value = cloudCurvature[i];
cloudSmoothness[i].ind = i;

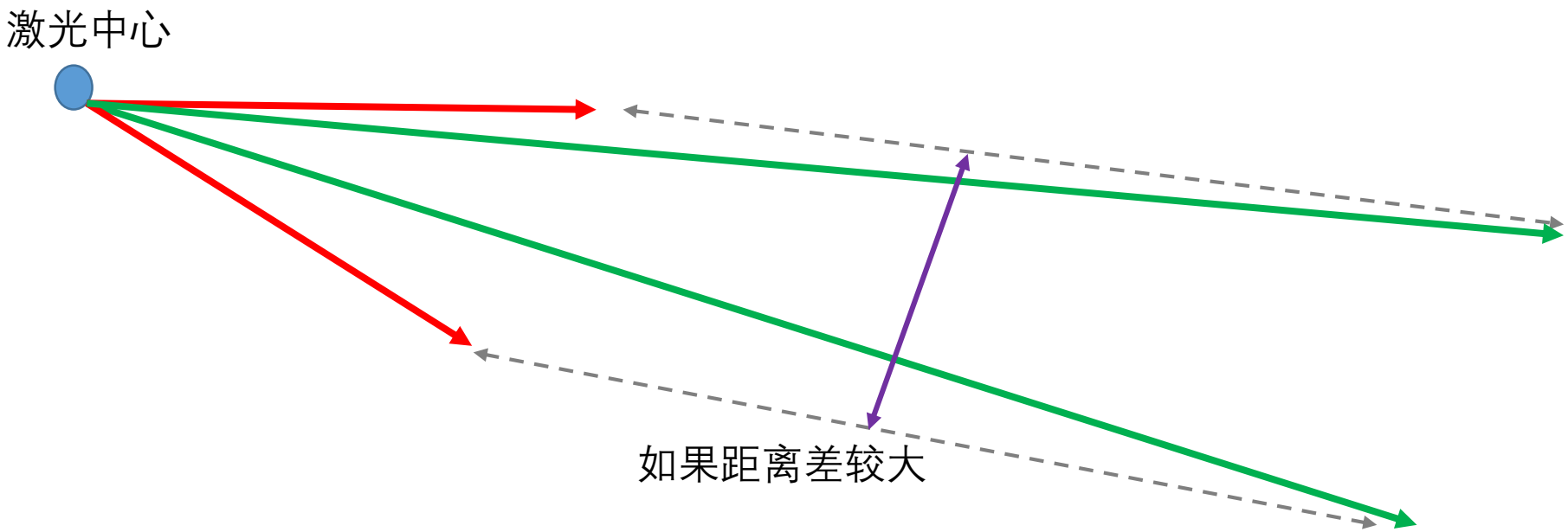
i	1	2	3	4	5	6
cloudCurvature	0.5	0.3	0.7	0.6	0.1	0.4
cloudLabel	-1	0	1	1	1	0
cloudNeighbor Picked	0	0	1	0	0	1

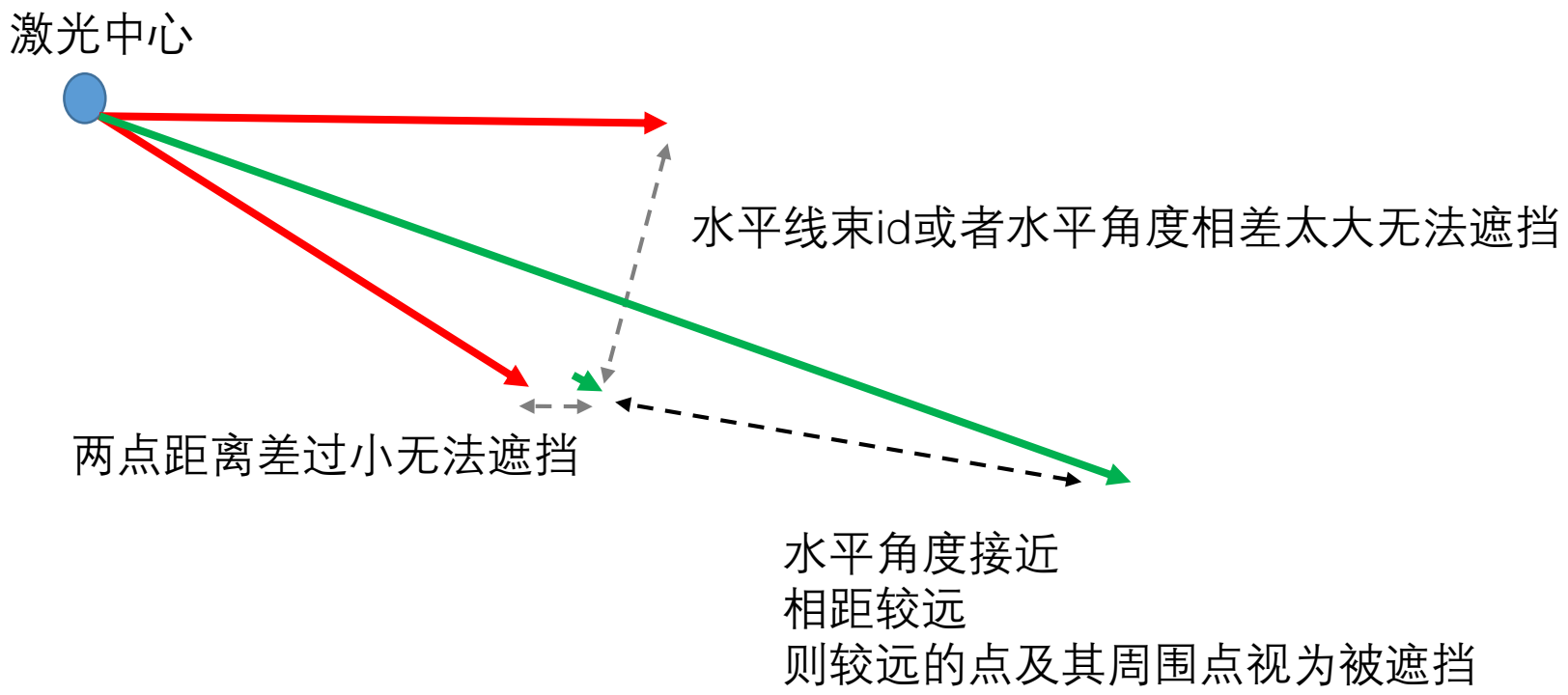
cloudSmoothness:按照曲率排序，i随曲率排序而变化，有i可关联到相关状态

Value cloudCurvature	0.5	0.3	0.7	0.6	0.1	0.4
Ind i	1	2	3	4	5	6

Value cloudCurvature	0.1	0.3	0.4	0.5	0.6	0.7
Ind i	5	2	6	1	4	3

前提是：点的水平角接近




激光中心
水平线束id或者水平角度相差太大无法遮挡
两点距离差过小无法遮挡
水平角度接近
相距较远
则较远的点及其周围点视为被遮挡

LIO-SAM中定义的message

Cloud Info：点云信息
//时间戳相关信息
Header header

int32[] startRingIndex
// 各个scan计算曲率的起点序号构成的数组;

int32[] endRingIndex
// 各个scan计算曲率的终点序号构成的数组;

int32[] pointColInd
// 点云中的点对应水平线束id
// 即点云映射的矩阵的列号

float32[] pointRange
// 点云中点到雷达中心距离

// 标记位，imu和里程计是否可进行运动补偿
int64 imuAvailable
int64 odomAvailable

// imu测量值得到的激光帧初始时刻姿态
float32 imuRollInit
float32 imuPitchInit
float32 imuYawInit

// imu预积分里程计得到的激光真初始时刻姿态
float32 initialGuessX
float32 initialGuessY
float32 initialGuessZ
float32 initialGuessRoll
float32 initialGuessPitch
float32 initialGuessYaw

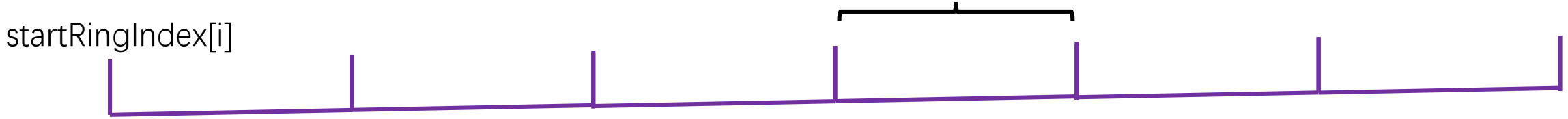
// 点云消息
sensor_msgs/PointCloud2 cloud_deskewed

// 角特征点云
sensor_msgs/PointCloud2 cloud_corner

// 面特征点云
sensor_msgs/PointCloud2 cloud_surface

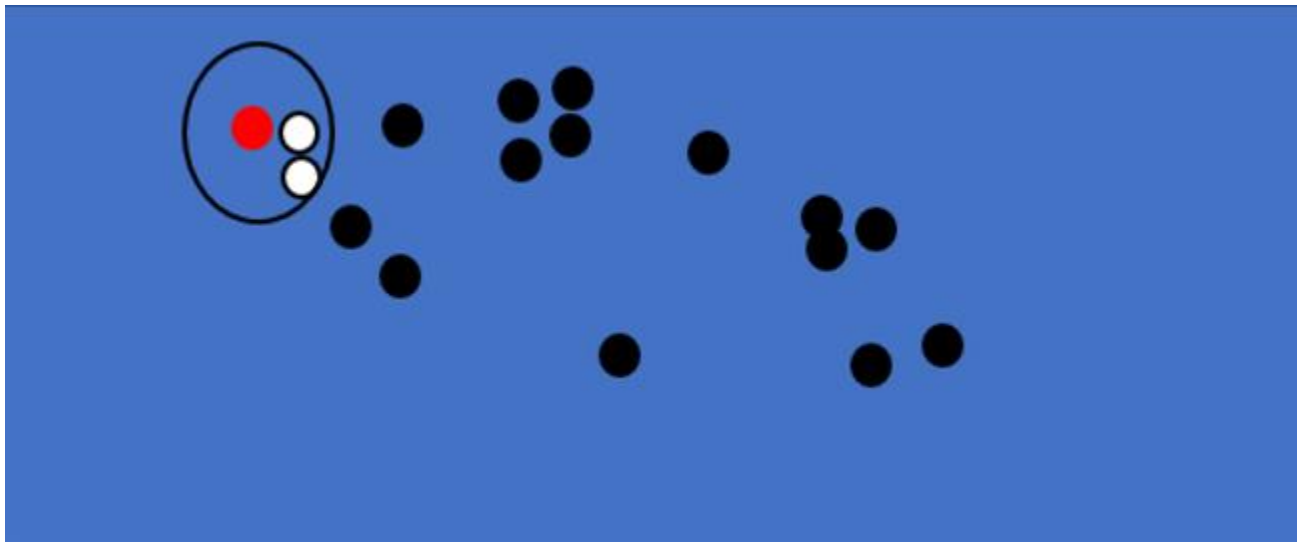
```
int sp = (cloudInfo.startRingIndex[i] * (6 - j) + cloudInfo.endRingIndex[i] * j) / 6;
       = cloudInfo.startRingIndex[i] + (cloudInfo.endRingIndex[i] - cloudInfo.startRingIndex[i])*j/6;
```

(cloudInfo.endRingIndex[i] - cloudInfo.startRingIndex[i])/6;




```
int ep = (cloudInfo.startRingIndex[i] * (5 - j) + cloudInfo.endRingIndex[i] * (j + 1)) / 6 - 1;
       = (cloudInfo.startRingIndex[i] * (6-(1 + j)) + cloudInfo.endRingIndex[i] * (j + 1)) / 6 - 1;
       = cloudInfo.startRingIndex[i] + (cloudInfo.endRingIndex[i] - cloudInfo.startRingIndex[i])*(j+1)/6 -1;
```

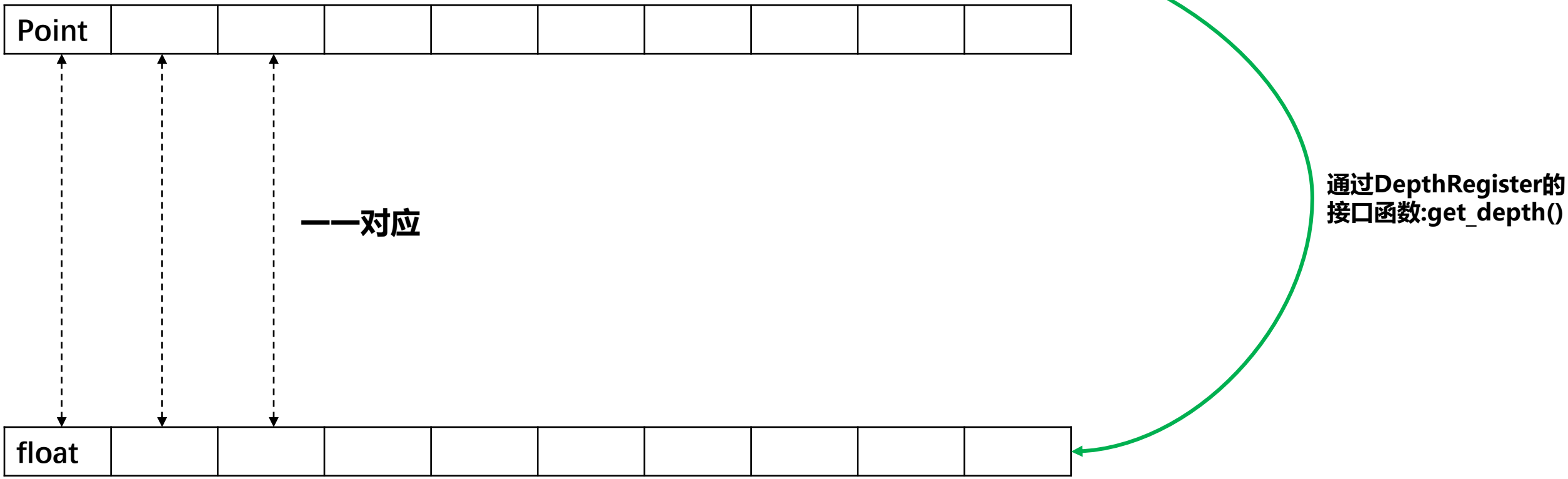
Visual_feature



深度关联

图像特征点集合： const vector<geometry_msgs::Point3>& **features_2d;**

PointXYZ: x、y、1



图像特征点对应的深度集合： sensor_msgs::ChannelFloat32 **depth_of_point;**

float: 初始值-1

Lidar_callback回调函数：获取到5s以内融合后的点云，为全局坐标系；

1、判断是否为空，若空则退出；

2、雷达点云坐标系对齐至图像坐标系；

3、深度关联；

3.1、将图像特征点归一化坐标映射至单位球面；

```
pcl::PointCloud<PointType>::Ptr features_3d_sphere(new pcl::PointCloud<PointType>());
for (int i = 0; i < (int)features_2d.size(); ++i)
{
    // normalize 2d feature to a unit sphere
    Eigen::Vector3f feature_cur(features_2d[i].x, features_2d[i].y, features_2d[i].z); // z always equal to 1
    feature_cur.normalize();
    // convert to ROS standard
    PointType p;
    p.x =  feature_cur(2);
    p.y = -feature_cur(0);
    p.z = -feature_cur(1);
    p.intensity = -1; // intensity will be used to save depth
    features_3d_sphere->push_back(p);
}
```

Lidar_callback回调函数：获取到5s以内融合后的点云，为全局坐标系；

1、判断是否为空，若空则退出；

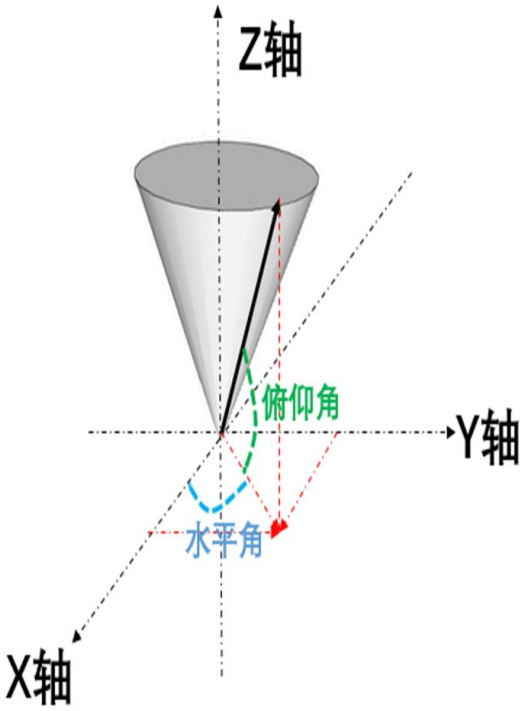
2、雷达点云坐标系对齐至图像坐标系；

3、深度关联；

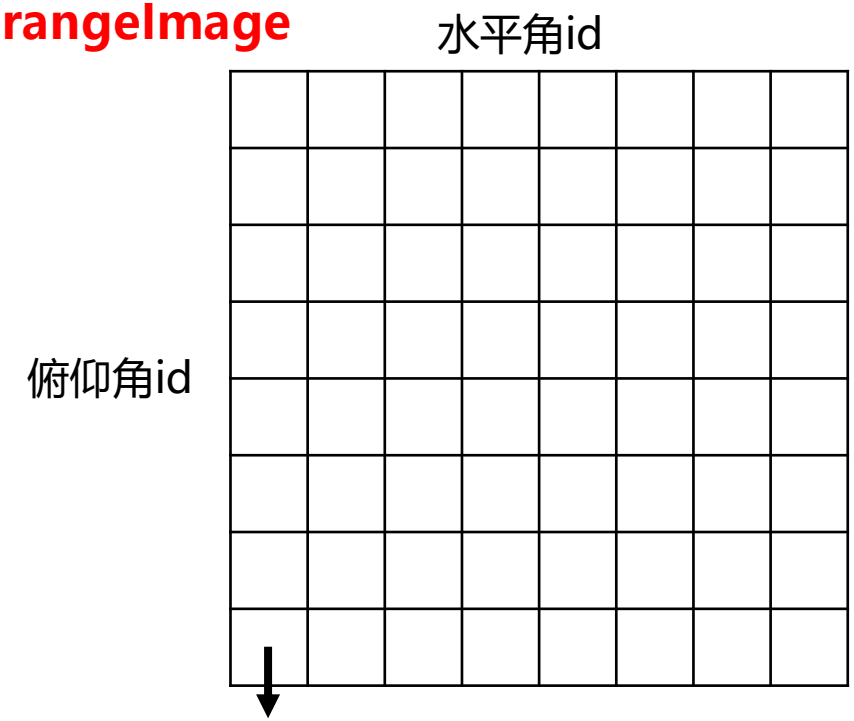
3.1、将图像特征点归一化坐标映射至单位球面；

3.2、雷达点云网格化，过滤相同区域重复的点，然后将其结果重建为点云形式；

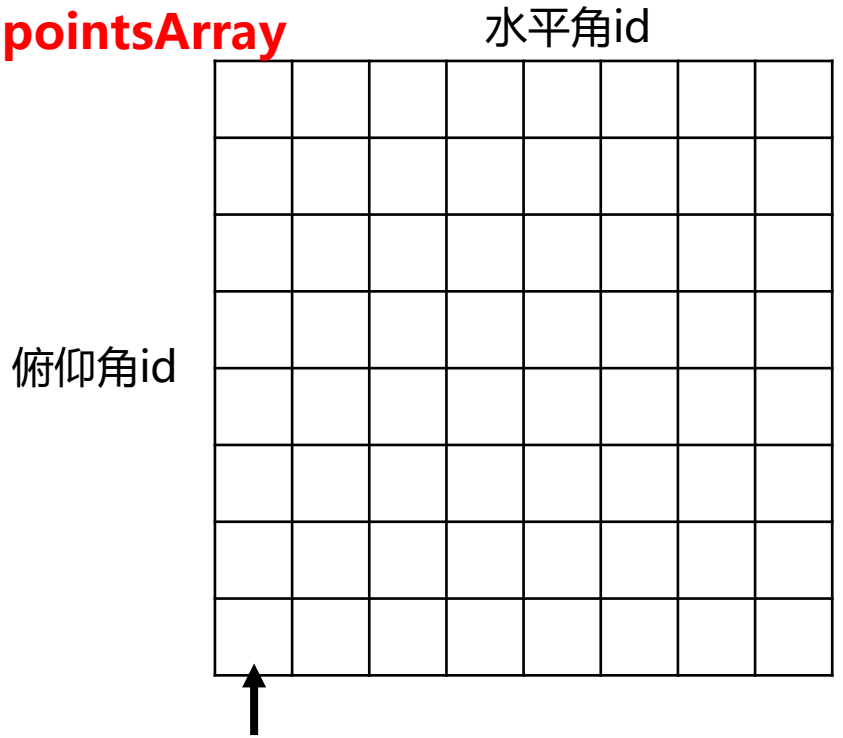
水平角180度范围，俯仰角180度范围，以0.5度为间隔形成一个360*360的网格；



rangeImage



pointsArray



Float:当前网格中点云中的点其距离最小 → Point:当前网格中点云其距离最小对应的点

Lidar_callback回调函数：获取到5s以内融合后的点云，为全局坐标系；

1、判断是否为空，若空则退出；

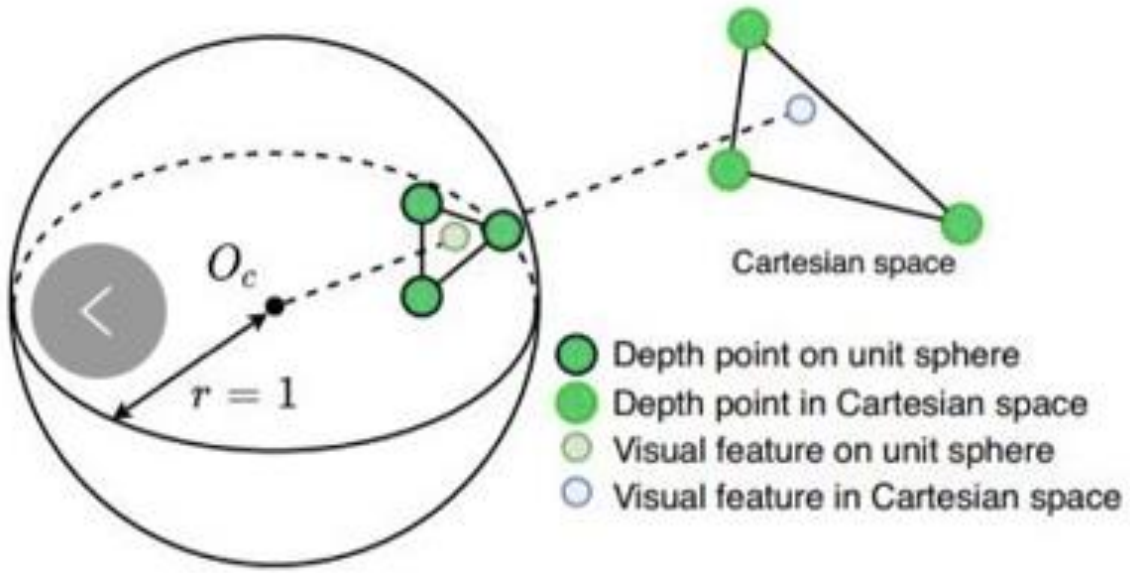
2、雷达点云坐标系对齐至图像坐标系，depth_cloud_local；

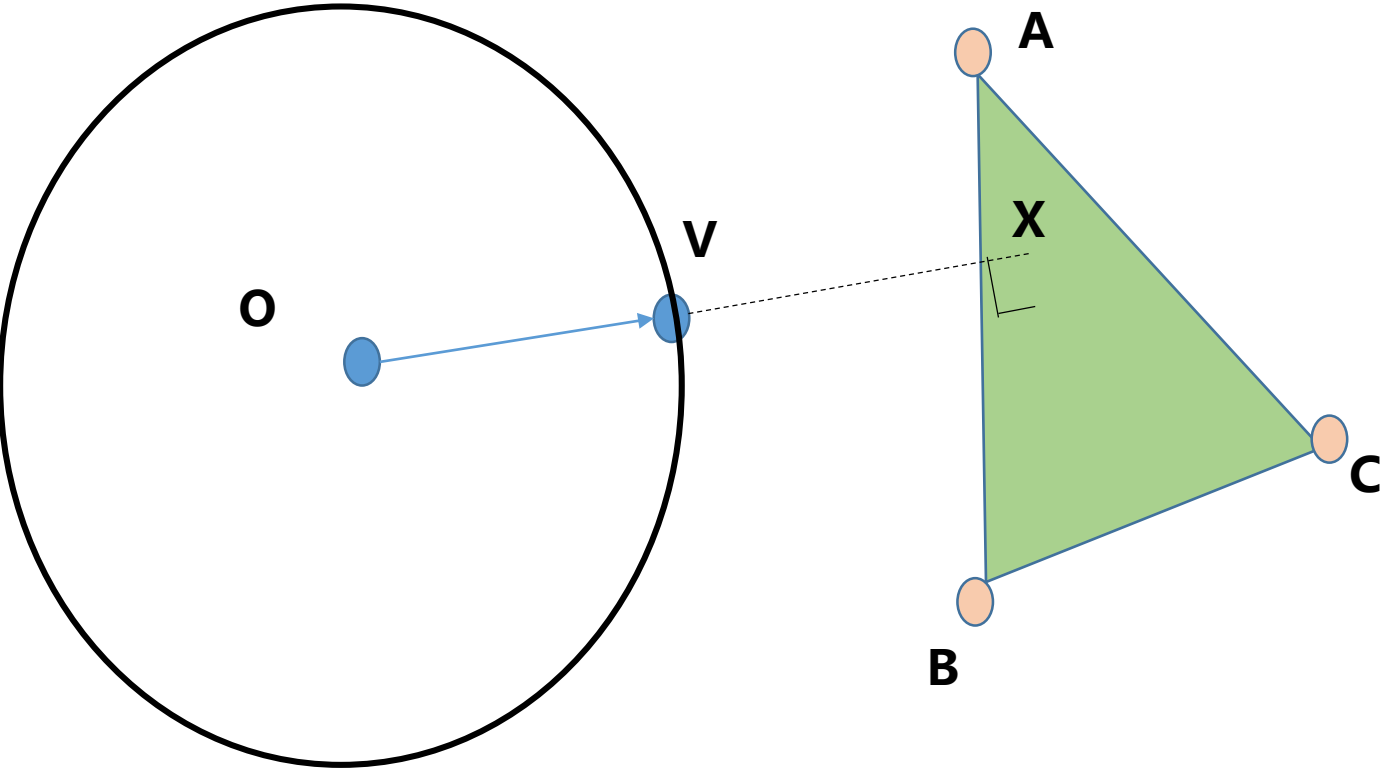
3、深度关联；

3.1、将图像特征点归一化坐标映射至单位球面；

3.2、雷达点云网格化，过滤相同区域重复的点，然后将其结果重新存入点depth_cloud_local；

3.3、将雷达点云归一化至单位球面，以该球面建立kd树，在树上搜索距离当前特征点最近的的三个点；





Eigen::Vector3f N = (A - B).cross(B - C);
表示平面ABC的法向量(n0,n1,n2）；

设平面ABC上任意一点为（x,y,z)
则平面的方程可表示为：
X-A表示平面ABC内任意向量，与法向量N垂直；
n0*(x-a1)+n2*(y-a2)+n3*(z-a3)=0;

设向量OV与平面ABC相交点为X，
由于向量OV与OX共起点且方向相同，
所以OX=t*OV;
因为OV在单位球面，长度为1，
所以t实际表示的是OX的长度；
将X带入平面方程：

n0*(t*v(0)-a0)+n1*(t*v(1)-a1)+n2*(t*v(2)-a2)=0

t*[n0*v(0)+n1*v(1)+n2*v(2)]=n0*a0+n1*a1+n2*a2

t= [n0*a0+n1*a1+n2*a2] /
*[n0*v(0)+n1*v(1)+n2*v(2)]

向量OV与平面ABC的交点可能在平面ABC以外，所以OX长度可能比OA、OB、OC都要小；