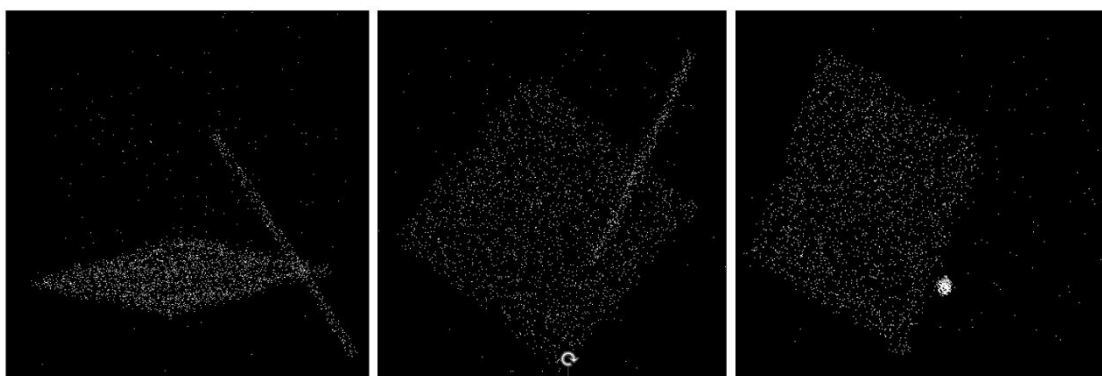


三维视觉与理解课程

实验四（大作业）

(1) 滤波并分割提取目标（占课程总成绩20%）

点云“大作业任务一.pcd”中包含一些属于“平面”的点，还包含一些属于“直线”的点，此外还存在一些干扰的



“噪点”，如下图所示。




请你设计一套算法，实现以下功能：

- 1、滤除给定点云中的噪点；
- 2、提取到属于“平面”的点，标为红色；提取属于“直线”的点，标为黄色；

需要提交的材料：（打成一个压缩包提交）

①实验的全部代码；

②实验的步骤记录,说明选用了哪些方法,提供必要的截图;

<p>①</p>  <p>原图</p>	<p>②</p>  <p>半径滤波</p> <p><code>nb_points=100,radius=0.8</code> <code>nb_points</code>: 指定在计算离群点时考虑的邻居点的最小数量。如果在指定半径范围内的邻居点数量少于此值,则当前点将被标记为离群点 <code>radius</code>: 指定用于确定邻居点的搜索半径。在给 定半径范围内搜索邻居点以确定每个点是否为离群 点</p>
<p>③</p>  <p>统计滤波</p>	<p><code>nb_neighbors=5,std_ratio=0.25</code> <code>nb_neighbors</code>: 最近k个点 <code>std_ratio</code>: 基于标准差的阈值,越小滤除点越多 原理: 去除分布稀疏的点,对于点云中的每个点,计算到最近k个点的平均距离,然后假设结果构成高斯分布,过滤平均距离大于标准差的点</p>

step1: 点云读取,染色为红色点云

```
file_name = "大作业任务一.pcd"  
pcd = o3d.io.read_point_cloud(file_name)  
pcd.paint_uniform_color([1,0,0])  
o3d.visualization.draw_geometries_with_editing([pcd])
```

step2: 半径滤波

原理: 去除分布稀疏的点,以每个点为中心建立给定半径的球体,移除球体中点的数量小于给定阈值的点

```
#半径滤波  
cl, ind = pcd.remove_radius_outlier(nb_points=100,radius=0.8)  
sor_cloud = pcd.select_by_index(ind)  
o3d.visualization.draw_geometries([sor_cloud], window_name="半径滤波")
```

step3: 统计学离散点群滤波

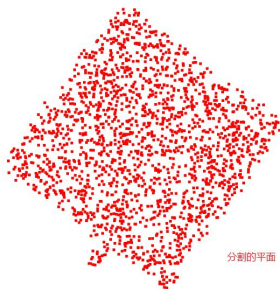
原理: 去除分布稀疏的点,对于点云中的每个点,计算到最

近 k 个点的平均距离，然后假设结果构成高斯分布，过滤平均距离大于标准差的点

#统计学离群点滤波

```
cl, ind2 = sor_cloud.remove_statistical_outlier(nb_neighbors=5, std_ratio=0.25)
sor_cloud2 = sor_cloud.select_by_index(ind2)
o3d.visualization.draw_geometries([sor_cloud2], window_name="统计滤波")
```

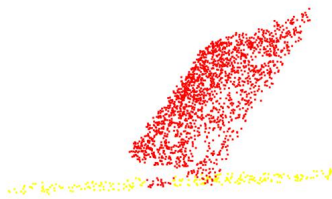
step4: 点云分割（重叠部分解决方案，最终结果在下面）



分割的平面



分割得到的直线

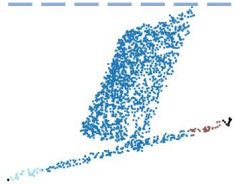


RANSAC平面分割

原理：假设点集中有一条直线 L ， L 外的点很少，均为噪声。随机选取两个点连成一条直线 $L \wedge h$ ，那么这条直线有可能就是 L ，也有可能是噪声连出来的莫名其妙的一条线。接下来，随机抽取点集中的一些点，如果随机抽取的大部分点都落在 L 附近，那么就说明 $L \wedge h$ 有很大的概率就是 L ；否则说明不太像是 L 。随着抽取出的直线越来越多，最后可以得到最接近 L 的直线，从而完成了对点集的分割。

输入参数：内点到平面模型的最大距离 d ，用于拟合平面的采样点数 n ，以及最大迭代次数 $nIter$

($d = 0.09, n = 3, nIter = 100$)



DBSCAN聚类

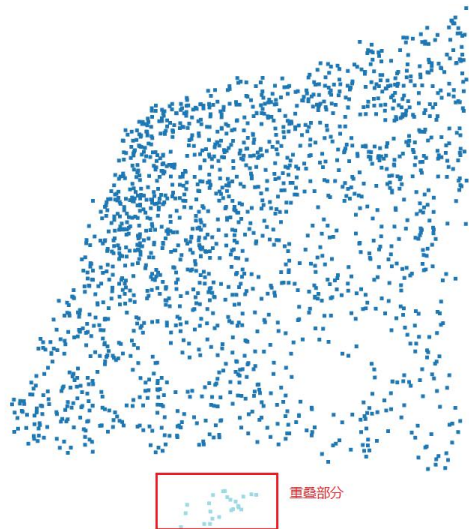
基于密度的噪声应用空间聚类

输入参数分别是 ϵ ，表示同一聚类中最大点间距； M 表示有效聚类的最小点数

$\epsilon = 0.15$, # 邻域距离
 $\min_points = 20$, # 最小点数

面临问题：重叠部分被分为了红色部分，如何解决？我的思路是先用 RANSAC 将红色部分提取出来（包含重叠部分），然后进一步使用 DBSCAN 聚为两类，实现红色平面和重叠部

分的分割。如下：



这样就得到了完整的平面，再将剩下的和黄色合并就行。

直接基于 RANSAC 平面分割（效果较好）：

```
#RANSAC平面分割
cl, ind3 = sor_cloud2.segment_plane(0.09, 3, 100)
sor_cloud3 = sor_cloud2.select_by_index(ind3)
0sor_cloud3 = sor_cloud2.select_by_index(ind3, invert=True)
0sor_cloud3.paint_uniform_color([1, 1, 0])
o3d.visualization.draw_geometries([sor_cloud3, 0sor_cloud3])
```

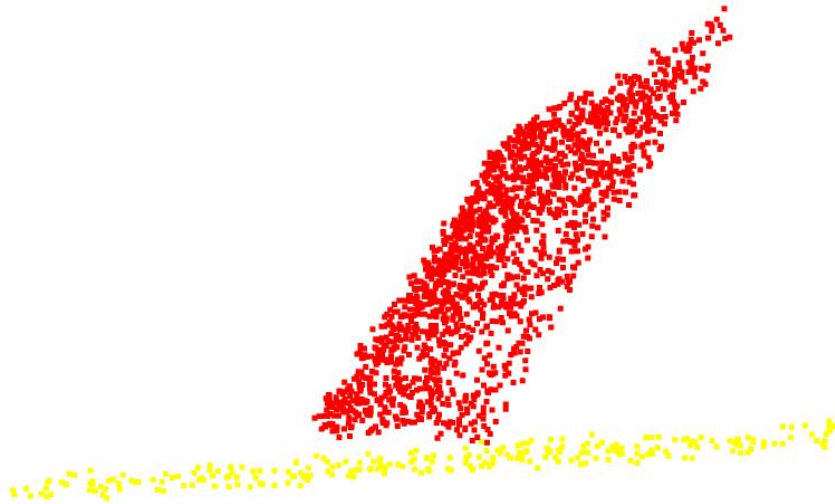
直接基于 DBSCAN 聚类：（效果差，直线部分密度不均匀识别成其他类的较多）

```
#DBSCAN聚类
# 设置为debug调试模式
with o3d.utility.VerboesityContextManager(o3d.utility.VerboesityLevel.Debug) as cm:
    # ----- 密度聚类 -----
    labels = np.array(sor_cloud2.cluster_dbscan(eps=0.15, min_points=20, print_progress=True))
    # 邻域距离
    # 最小点数
    # 是否在控制台中可视化进度条
max_label = labels.max()
# ----- 保存聚类结果 -----
for i in range(max_label + 1):
    ind = np.where(labels == i)[0]
    clusters_cloud = sor_cloud2.select_by_index(ind)
    # file_name = "Dbscan_cluster" + str(i+1) + ".pcd"
    # o3d.io.write_point_cloud(file_name, clusters_cloud)
# ----- 可视化聚类结果 -----
colors = plt.get_cmap("tab20")(labels / (max_label if max_label > 0 else 1))
colors[labels < 0] = 0
sor_cloud2.colors = o3d.utility.Vector3dVector(colors[:, :3])
o3d.visualization.draw_geometries([sor_cloud2], window_name="点云DBSCAN分割")
```

③提取后的平面点云文件、直线点云文件，分别命名为 plane.pcd 和 line.pcd。


```
o3d.io.write_point_cloud("plane.pcd", sor_cloud3)
o3d.io.write_point_cloud("line.pcd", 0sor_cloud3)
```

line.pcd	2024-06-19 10:14	PCD 文件	4 KB
plane.pcd	2024-06-19 10:14	PCD 文件	30 KB



(2) 配准并计算距离 (占课程总成绩 20%)

点云“大作业任务二 A.pcd”中是一个恐龙的点云，但比较稀疏。而点云“大作业任务二 B.pcd”是一个复杂环境的点云，包括很多恐龙，且点云密度较高。

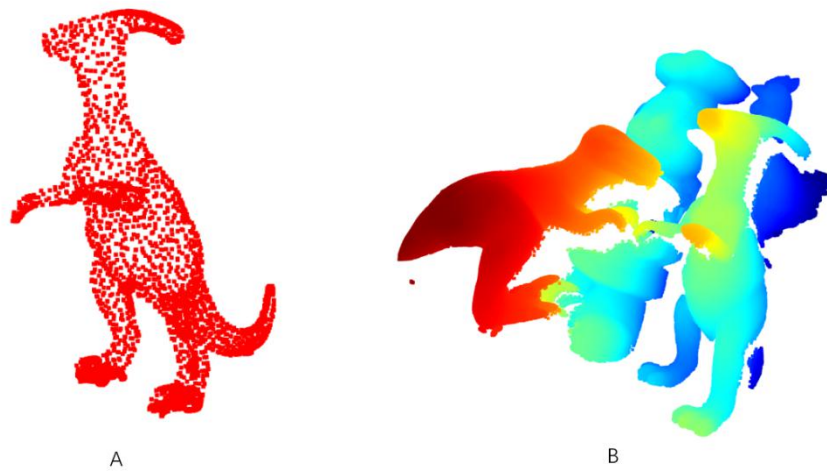
请你设计一套算法，将两个点云中相同的恐龙进行配准，针对配准后的点云，尝试计算该恐龙的身高(坐标单位是 cm)。

step1. 点云读取

```
pcd_A = o3d.io.read_point_cloud("大作业任务二A.pcd")
pcd_A.paint_uniform_color([1,0,0])

pcd_B = o3d.io.read_point_cloud("大作业任务二B.pcd")

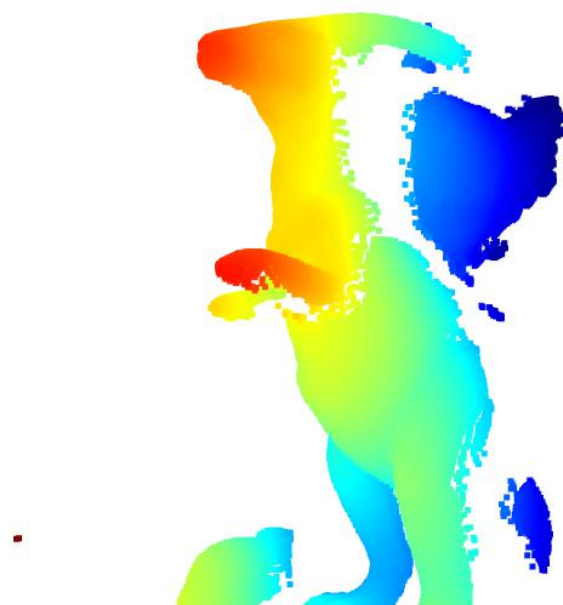
o3d.visualization.draw_geometries_with_editing([pcd_A])
o3d.visualization.draw_geometries_with_editing([pcd_B])
```



step2. 过滤掉其他杂恐龙，需要使用直通滤波&统计滤波(对 B 点云)

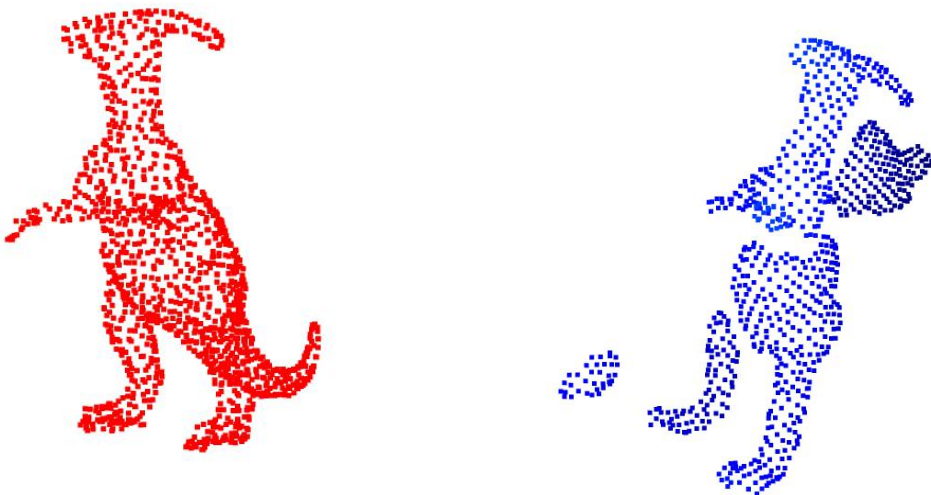
直通滤波，通过设定某个或某些轴（如 x 、 y 、 z 轴）的范围来筛选点云数据，只保留在指定范围内的点。直通滤波通常用于预处理步骤，去除明显不在感兴趣区域内的点。

统计学滤波，用于去除离群点（噪声点）。这种方法基于每个点在其邻域内的统计特性，通过计算每个点到其邻居点的距离并利用这些距离的统计信息（如均值和标准差）来识别和删除离群点。下面是两次滤波后的 B 点云



step3. 统一密度便于配准，使用体素均匀下采样（对 A,B 点云）

体素下采样会将点云划分为一个个小的体素网格，并用每个体素内点的平均值代替原始点，从而实现降噪。**体素均匀下采样将两只恐龙的密度调整一致（大差不差），以便于后续配准查找对应点（FPFH）。**



点云 A 密度为 $\text{density} = 4.528717440650566$

点云 B 密度为 $\text{density} = 3.6011470178898586$

```
voxel_size = 6 # 根据需要调整体素大小
downPcd_A = pcd_A.voxel_down_sample(voxel_size)
downPcd_B = sor_cloud.voxel_down_sample(voxel_size)
o3d.visualization.draw_geometries([downPcd_A, downPcd_B], window_name="体素均匀下采样")
```

均匀采样

```
point = np.asarray(downPcd_A.points) # 获取点坐标
kdtree = o3d.geometry.KDTreeFlann(downPcd_A) # 建立KD树索引
point_size = point.shape[0] # 获取点的个数
dd = np.zeros(point_size)
for i in range(point_size):
    [_, idx, dis] = kdtree.search_knn_vector_3d(point[i], 2)
    dd[i] = dis[1] # 获取到最近邻点的距离平方
density = np.mean(np.sqrt(dd)) # 计算平均密度
print("点云A密度为 density=", density)

point = np.asarray(downPcd_B.points) # 获取点坐标
kdtree = o3d.geometry.KDTreeFlann(downPcd_B) # 建立KD树索引
point_size = point.shape[0] # 获取点的个数
dd = np.zeros(point_size)
for i in range(point_size):
    [_, idx, dis] = kdtree.search_knn_vector_3d(point[i], 2)
    dd[i] = dis[1] # 获取到最近邻点的距离平方
density = np.mean(np.sqrt(dd)) # 计算平均密度
print("点云B密度为 density=", density)
```

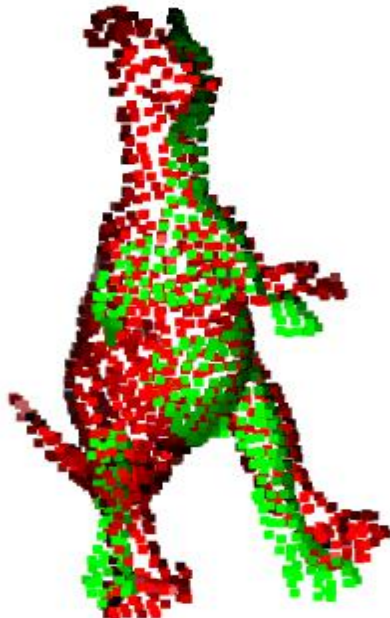
密度计算

step5. 点云配准

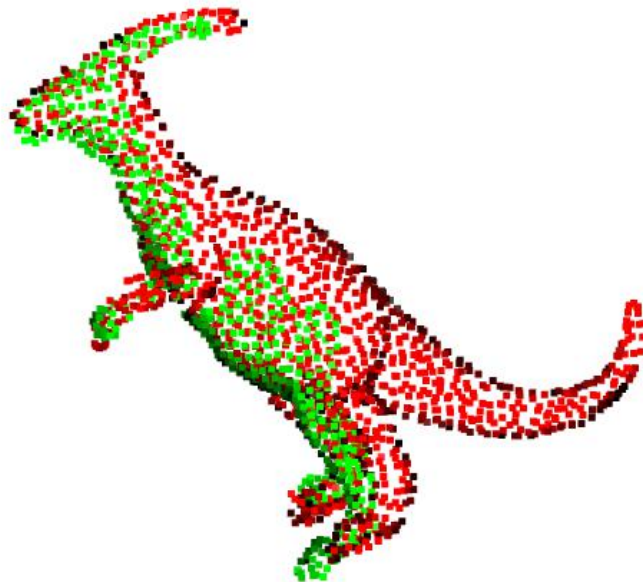
FPFH: 快速点特征直方图, PFH 是一种辛纳希丰富的局部特征, 对于一个有 n 个点的点云数据, PFH 的复杂度是 $O(nk^2)$, 而 FPFH 能够将复杂度降到 $O(nk)$, 同时保留 PFH 的大部分鉴别能力和特征。

```
# ----- 传入点云数据, 计算FPFH -----  
2 用法  
def FPFH_Compute(pcd):  
    radius_normal = 10 # kdtree 参数, 用于估计法线的半径,  
    print(":: Estimate normal with search radius %.3f." % radius_normal)  
    pcd.estimate_normals(  
        o3d.geometry.KDTreeSearchParamHybrid(radius=radius_normal, max_nn=50))  
# 估计法线的1个参数, 使用混合型的kdtree, 半径内取最多30个邻居  
    radius_feature = 10 # kdtree 参数, 用于估计FPFH特征的半径  
    print(":: Compute FPFH feature with search radius %.3f." % radius_feature)  
    pcd_fpfh = o3d.pipelines.registration.compute_fpfh_feature(pcd,  
        o3d.geometry.KDTreeSearchParamHybrid(radius=radius_feature, max_nn=50)) # 计算FPFH特征, 搜索方法kdtree  
    return pcd_fpfh # 返回FPFH特征
```

RANSAC, 一种迭代方法, 用于从包含大量噪声的样本中估计模型参数。通过反复从数据集中随机选取子集, 拟合模型, 并评估模型的可靠性, RANSAC 能够有效地从包含离群点的数据中提取出可靠的模型。下图是通过 FPFH 提取特征点+RANSAC 算法得到的粗配准图:

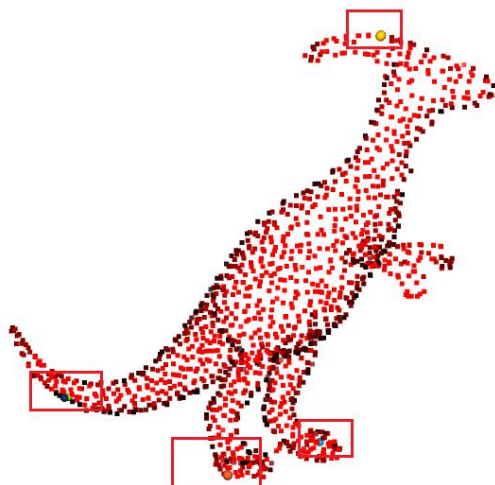


ICP, 在找到两个点云之间的最佳刚体变换(如旋转和平移), 使它们尽可能对齐。ICP 算法通过迭代优化来逐步改进配准结果, 直到满足收敛条件为止。下图是 ICP 精配准得到的图:



step6. 身高计算

运行 check.py 代码选择最高的顶点, 然后在底部选择三个点作为平面, 将参数带入点到平面距离计算函数中得到空得到恐龙的身高。



```
[Open3D INFO] Picked point #962 (-19., 1.3e+02, -6.2e+02) to add in queue.  
[Open3D INFO] No point has been picked.  
[Open3D INFO] No point has been picked.  
[Open3D INFO] Picked point #1353 (-2.2e+02, 29., -6.1e+02) to add in queue.  
[Open3D INFO] No point has been picked.  
[Open3D INFO] No point has been picked.  
[Open3D INFO] No point has been picked.  
[Open3D INFO] No point has been picked.  
[Open3D INFO] No point has been picked.  
[Open3D INFO] No point has been picked.  
[Open3D INFO] No point has been picked.  
[Open3D INFO] Picked point #21 (-2.5e+02, 1.2e+02, -6.6e+02) to add in queue.  
[Open3D INFO] Picked point #1062 (-2.2e+02, 26., -7.7e+02) to add in queue.  
[Open3D INFO] No point has been picked.  
[Open3D INFO] No point has been picked.  
[Open3D INFO] Picked point #1038 (-2.2e+02, 31., -7.7e+02) to add in queue.
```

```
def dist(point,a,b,c,d):
    """
    计算点到平面的距离
    :param point: 平面外面一点
    :param a: 平面方程系数a
    :param b: 平面方程系数b
    :param c: 平面方程系数c
    :param d: 平面方程系数d
    :return: 点到平面的距离
    """
    dis = abs(a*point[0] + b*point[1] + c*point[2] - d)/np.sqrt(a **2 + b**2 + c**2)
    return dis

def plane_equation_from_points(p1, p2, p3):
    # 计算向量
    v1 = np.array(p2) - np.array(p1)
    v2 = np.array(p3) - np.array(p1)

    # 计算叉积
    normal_vector = np.cross(v1, v2)
    a, b, c = normal_vector

    # 计算d
    d = -np.dot(normal_vector, p1)

    return a, b, c, d

a, b, c, d = plane_equation_from_points(
    np.array(new_pcd.points[40]),
    np.array(new_pcd.points[1353]),
    np.array(new_pcd.points[1009]))
print("平面系数: %d %d %d %d",a,b,c,d)
print("恐龙身高: ",dist(new_pcd.points[205],a,b,c,d))
```

恐龙身高: **221.70313512145609**

身高计算方法提示：可尝试使用 PCA 主方向方法，找到恐龙身高的方向，计算恐龙点云在该方向上的最大跨度；或通过恐龙的脚，构建一个“地平面”，计算点到平面的距离。

点到平面距离计算方法：

```
def dist(point, a, b, c, d):
    """
    计算点到平面的距离
    :param point: 平面外一点
    :param a: 平面方程系数a
    :param b: 平面方程系数b
    :param c: 平面方程系数c
    :param d: 平面方程系数d
    :return: 点到平面的距离
    """
    dis = abs(a * point[0] + b * point[1] + c *
              point[2] - d) / np.sqrt(a ** 2 + b ** 2 + c ** 2)
    return dis
```

*尽可能取得更好的效果，但不要担心因为结果不够好而得不到好的成绩，体现你对解决该问题的思考即可。

需要提交的材料：（打成一个压缩包提交）

- ①实验的全部代码；
- ②实验的步骤记录，说明选用了哪些方法，提供必要的截图；
- ③配准后的恐龙点云文件、恐龙的身高计算结果。