

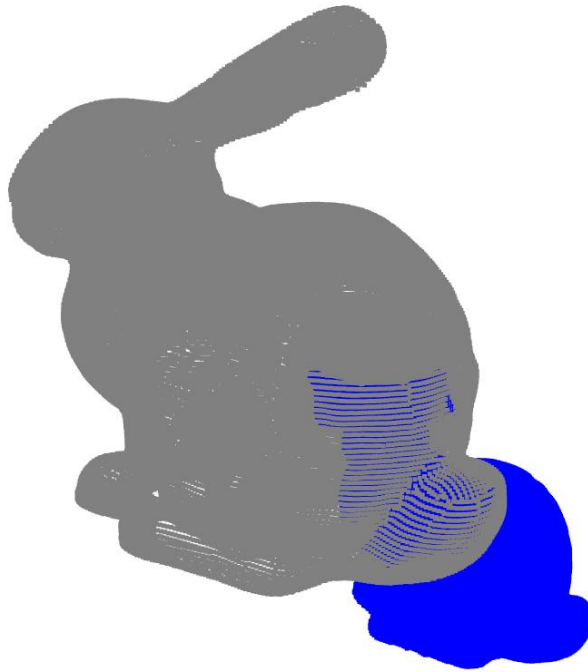
实验二报告【框架】

1、点云的平移和旋转

旋转平移关键代码：

```
Rotate = pcd_affine.get_rotation_matrix_from_xyz(rotation=[0, 0, np.pi]) # z 轴顺时针  
转 180°  
Translate = np.array([-0.1, 0, 0]).reshape(3, 1) # x 轴平移 0.1(向右)
```

平移旋转后对比结果：



旋转矩阵：

```
[[-1.00000000e+00 -1.2246468e-16  0.00000000e+00 -1.0000000e-01]  
 [ 1.2246468e-16 -1.0000000e+00  0.00000000e+00  0.0000000e+00]  
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.0000000e+00]  
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -2.0000000e+00]]
```

2、点云滤波

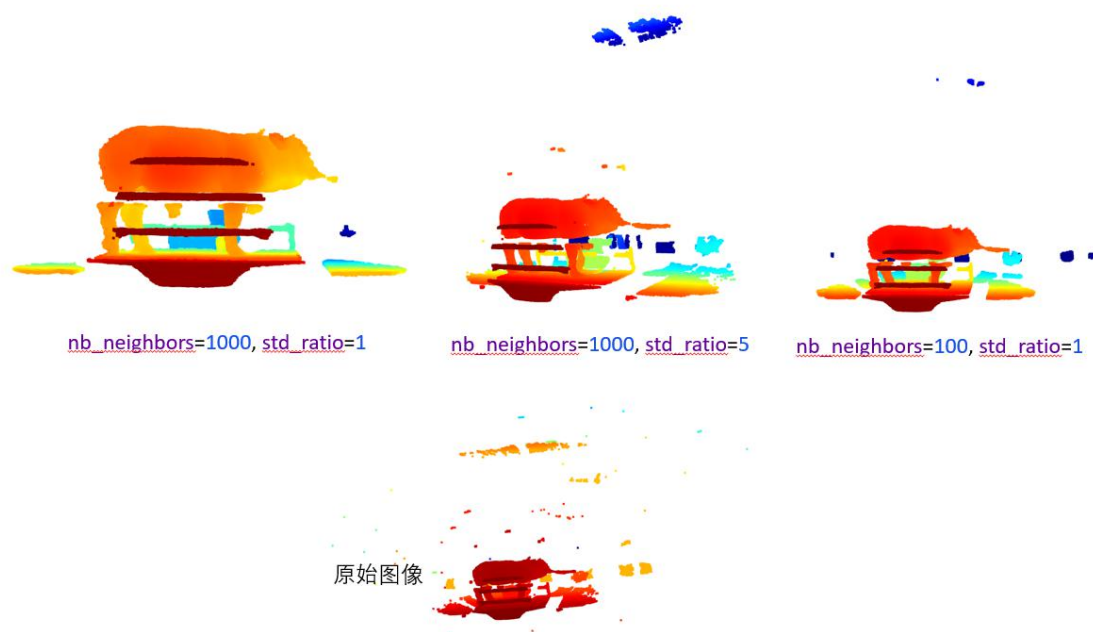
滤波过程概述：（含关键参数、中间过程截图，应包含三种滤波方法）

(1) 统计学滤波

参数含义：

Nb_neighbors 计算平均距离的邻域点数量

Std_ratio: 平均距离标准差阈值，越小滤波效果越明显。



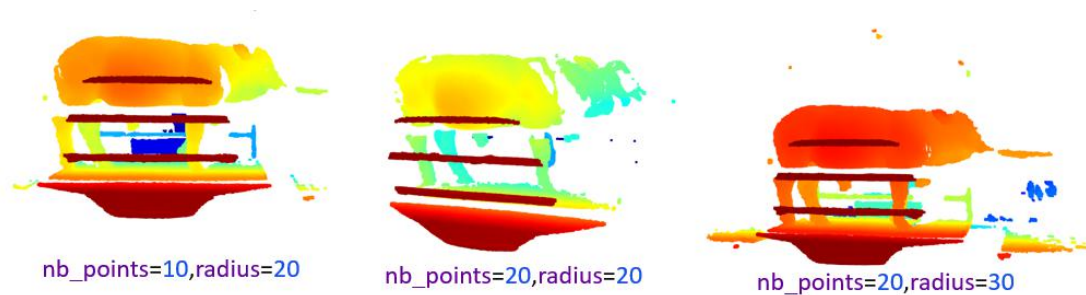
最佳参数 1: 1000, 参数 2: 1

效果：去掉了大部分离散的点，如上图最左边所示

(2) 半径滤波

参数含义： nb_points: 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。

radius: 指定用于确定邻居点的搜索半径。



效果：通过半径滤波，过滤掉了大部分不集中的点，左图效果较好

(3) 直通滤波

```
# 直通滤波

def pass_through_filter(pcd, axis, axis_min, axis_max):

    # 获取点云数据的 Numpy 数组

    points = np.asarray(pcd.points)

    # 根据指定的坐标范围筛选点云

    if axis == 'x':

        mask = (points[:, 0] > axis_min) & (points[:, 0] < axis_max)

    elif axis == 'y':

        mask = (points[:, 1] > axis_min) & (points[:, 1] < axis_max)
```

```
elif axis == 'z':

    mask = (points[:, 2] > axis_min) & (points[:, 2] < axis_max)

else:

    raise ValueError("Invalid axis. Axis must be 'x', 'y', or 'z'.")

# 创建新的点云对象

filtered_pcd = o3d.geometry.PointCloud()

filtered_pcd.points = o3d.utility.Vector3dVector(points[mask])


# 如果点云包含颜色信息，同样进行筛选

if pcd.has_colors():

    colors = np.asarray(pcd.colors)

    filtered_pcd.colors = o3d.utility.Vector3dVector(colors[mask])


# 如果点云包含法线信息，同样进行筛选

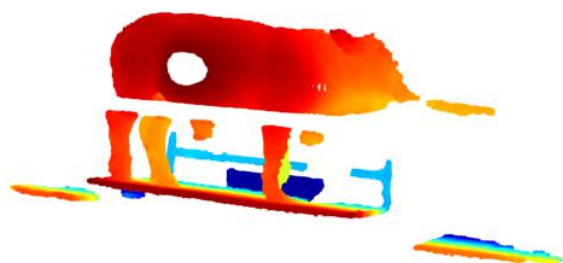
if pcd.has_normals():

    normals = np.asarray(pcd.normals)

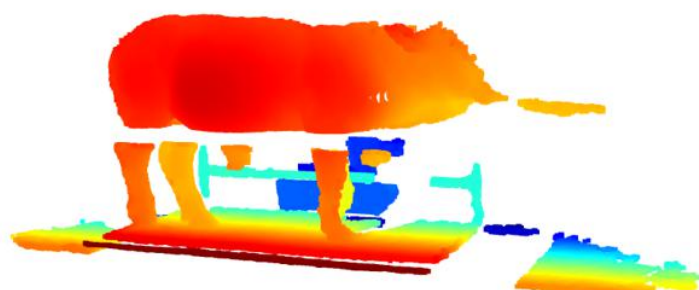
    filtered_pcd.normals = o3d.utility.Vector3dVector(normals[mask])


return filtered_pcd
```

最终滤波结果截图：




n1 = 4800
n2 = 5800



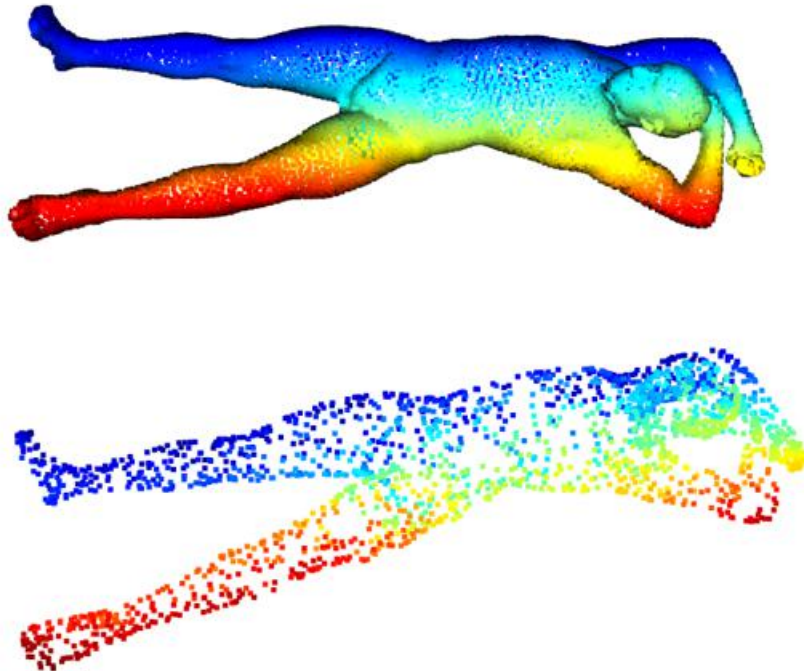
n1 = 3800
n2 = 5800

另请附 Q2.pcd 文件。

 Q2.pcd

3、点云的抽稀

抽稀前后对比截图：



原始点云点的个数为： 12500
下采样后点的个数为： 2051

简要描述曲率抽稀原理：

通过保留高曲率区域更多的点来减少点云数据的数量，同时尽量保持点云的几何特征。

大致过程：

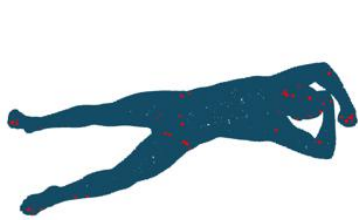
- 加载点云数据。
- 计算每个点的法向量。
- 基于法向量计算每个点的曲率。
- 根据曲率将点云分为高曲率区域和低曲率区

域。

- 分别对高曲率和低曲率区域进行均匀下采样。
- 合并下采样后的点云并进行可视化

4、点云关键点提取

(1) 使用代码 `key_point.py` 提取点云 `body.pcd` 中的 ISS 关键点，将结果图进行截图，并说明提取到了几个关键点



salient_radius=0.05

Extract PointCloud with 134 points.



salient_radius=0.1

Extract PointCloud with 78 points.



salient_radius=0.01

Extract PointCloud with 323 points.

确定邻域搜索的半径，主要用于计算点云中每个点的局部几何结构。这个半径值越大，考虑的邻域范围越大，提取到的关键点会更加全局化。



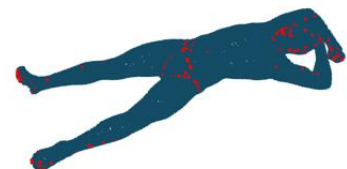
non_max_radius=0.1

Extract PointCloud with 19 points.



non_max_radius=0.05

Extract PointCloud with 55 points.

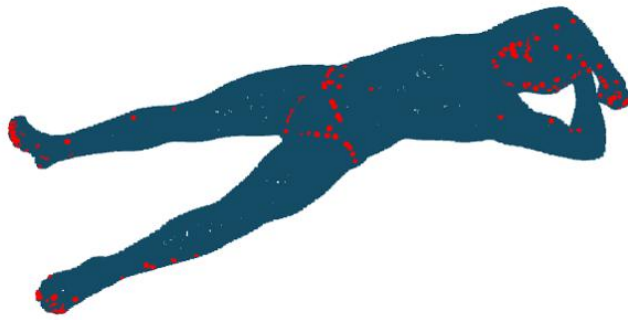


non_max_radius=0.01

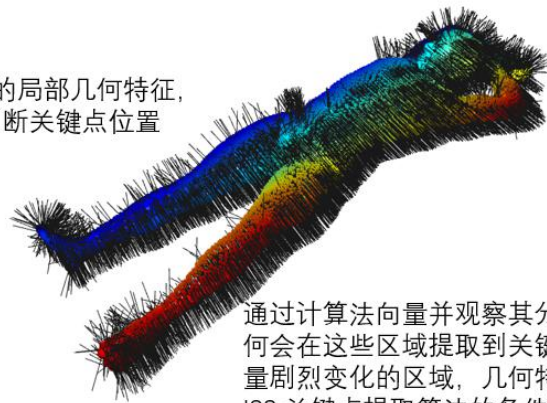
Extract PointCloud with 323 points.

用于非极大值抑制步骤。该半径值确定了在邻域内进行极大值抑制的范围，确保在指定范围内只有一个极大值被保留，其他极大值被抑制

(2) 使用 Normal.py 程序，为 body.pcd 点云中的所有点添加法向量。基于法向量的分布特征，简要解释在问题(1)中为何会提取到这些关键点？



ISS 关键点提取算法主要依据点云中每个点的局部几何特征，通过计算邻域点的协方差矩阵的特征值来判断关键点位置



通过计算法向量并观察其分布特征，可以理解为何会在这些区域提取到关键点。这是因为在法向量剧烈变化的区域，几何特征较为明显，满足 ISS 关键点提取算法的条件