# 实习汇报

## 课题：3D效果模型的优化:

1.安装Pycharm ,以及python环境以及CloudCompare以及下载数据集

2 .安装annaconda 环境

3.安装Open3D环境

4.3D模型的读取、精细化、平滑化+均值滤波、顶点聚类算法、曲面重建算法代码调试

5.PointNet++介绍，环境部署（pytorch，torchvision..），代码训练（分类和分割）、结果可视化

**Open3D**是一个开源的3D计算库，用于处理、分析和可视化三维数据。它提供了一系列用于处理点云、网格、图像和其他与3D数据相关的功能。Open3D的目标是为计算机视觉、机器人学、虚拟现实、增强现实等领域提供强大的工具，帮助研究人员和开发人员处理和分析三维数据。

Open3D的一些主要功能包括：

点云处理：Open3D提供了各种点云数据处理功能，包括滤波、采样、配准、分割等，可以用于从3D传感器（如激光雷达）获取的点云数据中提取有用信息。

网格处理：它支持网格数据的操作，例如表面重建、网格滤波、网格配准等。这对于处理3D扫描数据或CAD模型非常有用。

可视化：Open3D可以用于将3D数据可视化，帮助用户更好地理解和分析数据。它支持点云和网格的渲染，以及交互式的3D可视化界面。

深度学习集成：Open3D与深度学习框架（如PyTorch和TensorFlow）集成，使用户能够将深度学习模型应用于3D数据。

3D数据I/O：Open3D支持多种3D数据格式的读取和写入，包括PLY、OBJ、OFF等。

应用领域包括但不限于：

计算机视觉：在物体检测、物体识别和场景分析中，3D数据可以为计算机视觉任务提供更丰富的信息。Open3D可以用于处理和分析这些数据。

机器人学：在机器人感知和导航中，3D传感器可以为机器人提供环境的三维信息。Open3D可用于处理机器人获取的点云数据，从而帮助机器人做出更好的决策。

**增强现实和虚拟现实**：在AR和VR应用中，3D数据是创建逼真体验的关键。Open3D可以用于将真实世界的3D数据集成到虚拟环境中，或者将虚拟对象与现实世界进行融合。

地图构建与导航：在自动驾驶、无人机导航等领域，Open3D可以用于处理激光雷达等传感器获取的地图数据，从而帮助实现精确的定位和导航。

总之，Open3D是一个功能强大的3D计算库，适用于许多需要处理、分析和可视化三维数据的应用领域。

读取和加载3D数据是一个关键的步骤，它涉及将采集的点云数据从文件或传感器中提取出来，为后续的分析和处理做准备。Open3D作为一个广泛使用的点云处理库，提供了灵活的工具和函数来读取不同格式的3D数据。以下是关于3D数据读取的整体情况：

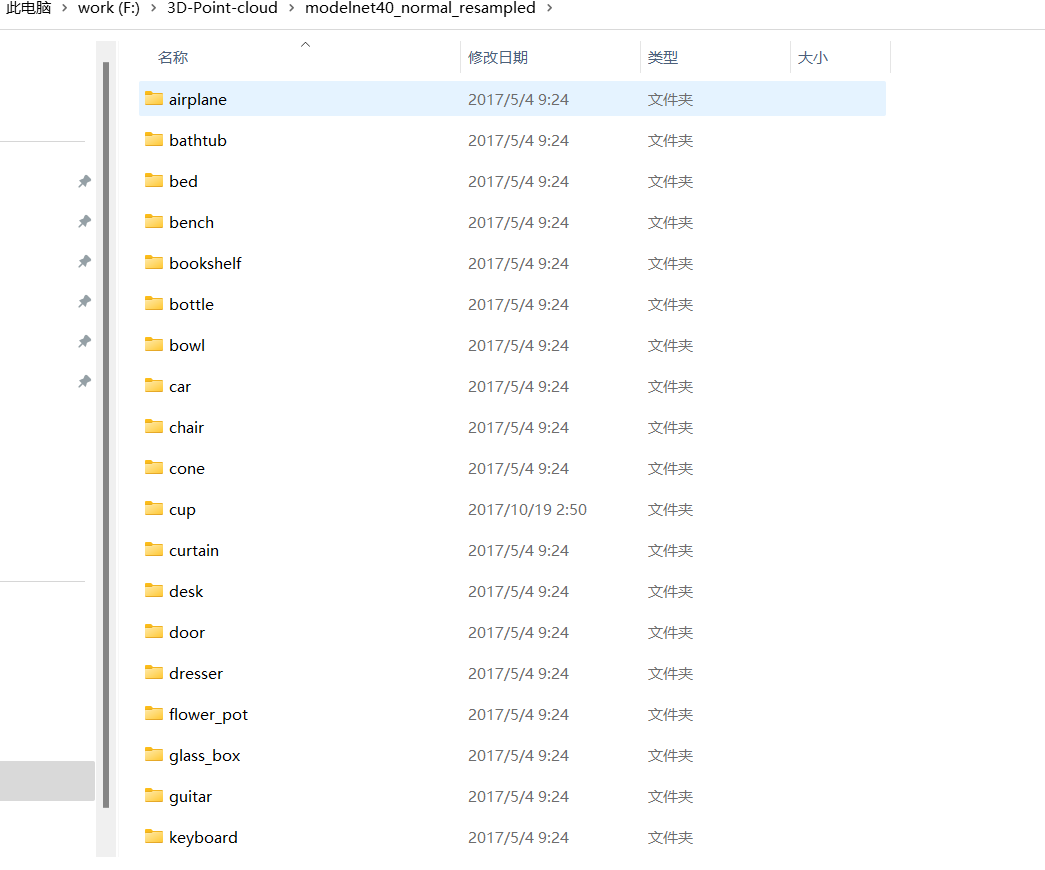
文件格式支持： Open3D支持多种常见的3D点云数据文件格式，包括PLY、PCD、XYZ等。这些格式能够存储点的坐标、颜色和法线等信息。

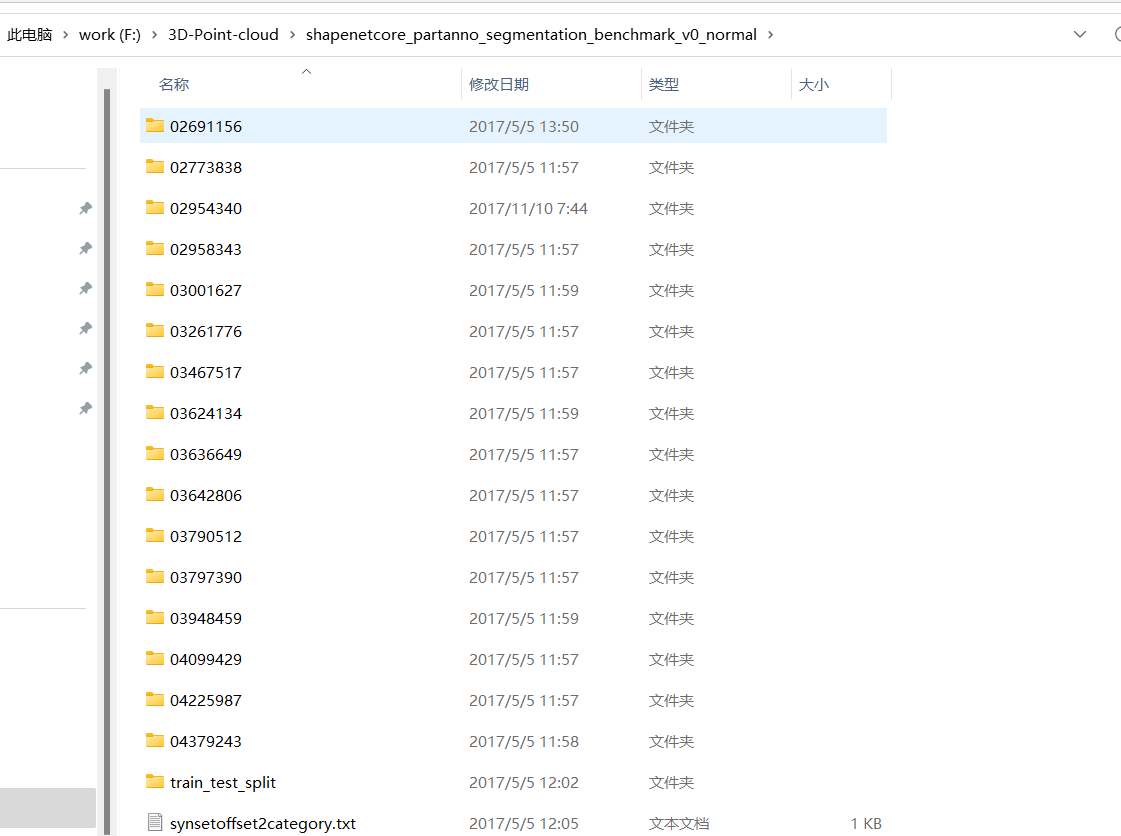
**点云数据来源**： 3D点云数据可以来自不同的来源，如激光扫描、摄像头捕捉、深度传感器等。Open3D提供了函数来从这些来源中读取数据。

数据加载方法： 使用Open3D，你可以通过简单的一行代码来加载点云数据。例如，使用read\_point\_cloud函数可以从文件中读取点云数据。

数据预处理： 在加载数据后，Open3D也提供了数据预处理的工具，如点云降采样、去噪、变换等。这些操作有助于优化数据的质量和效率。

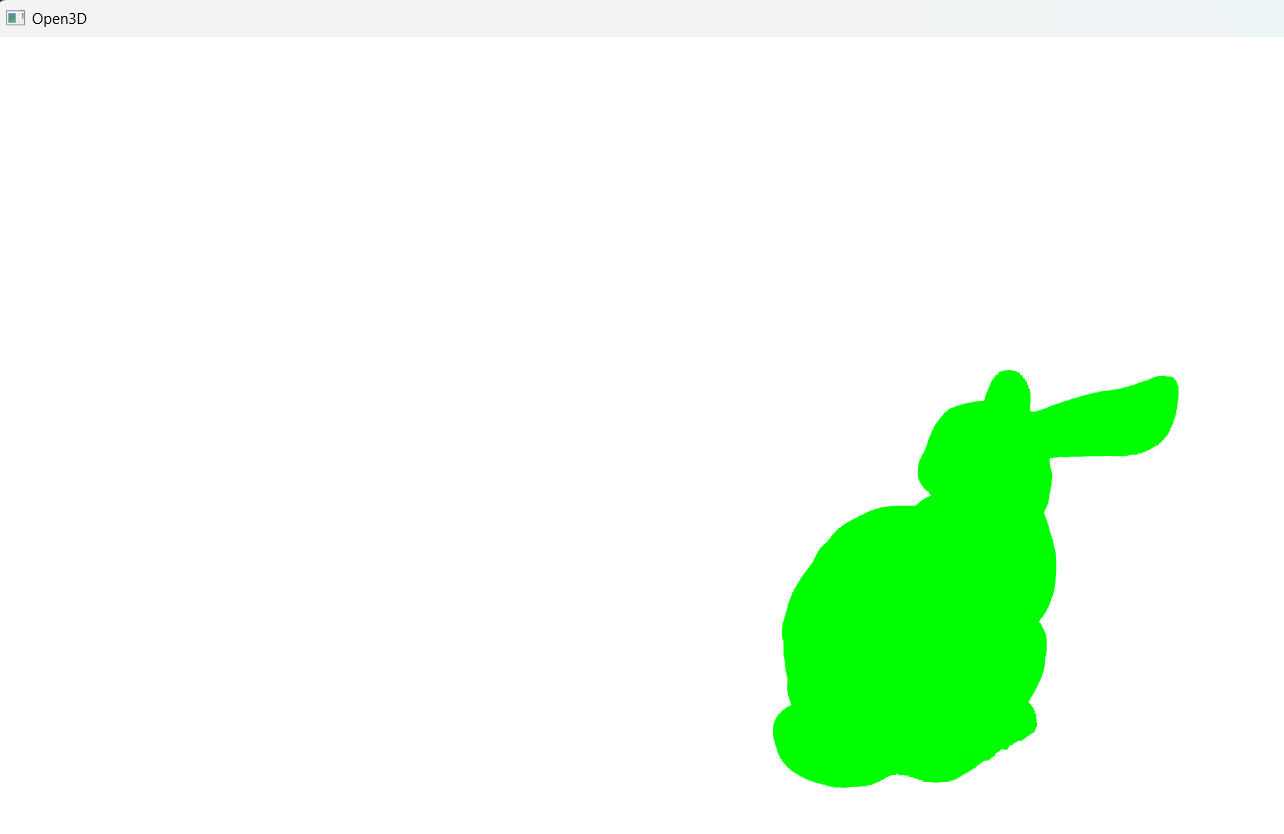
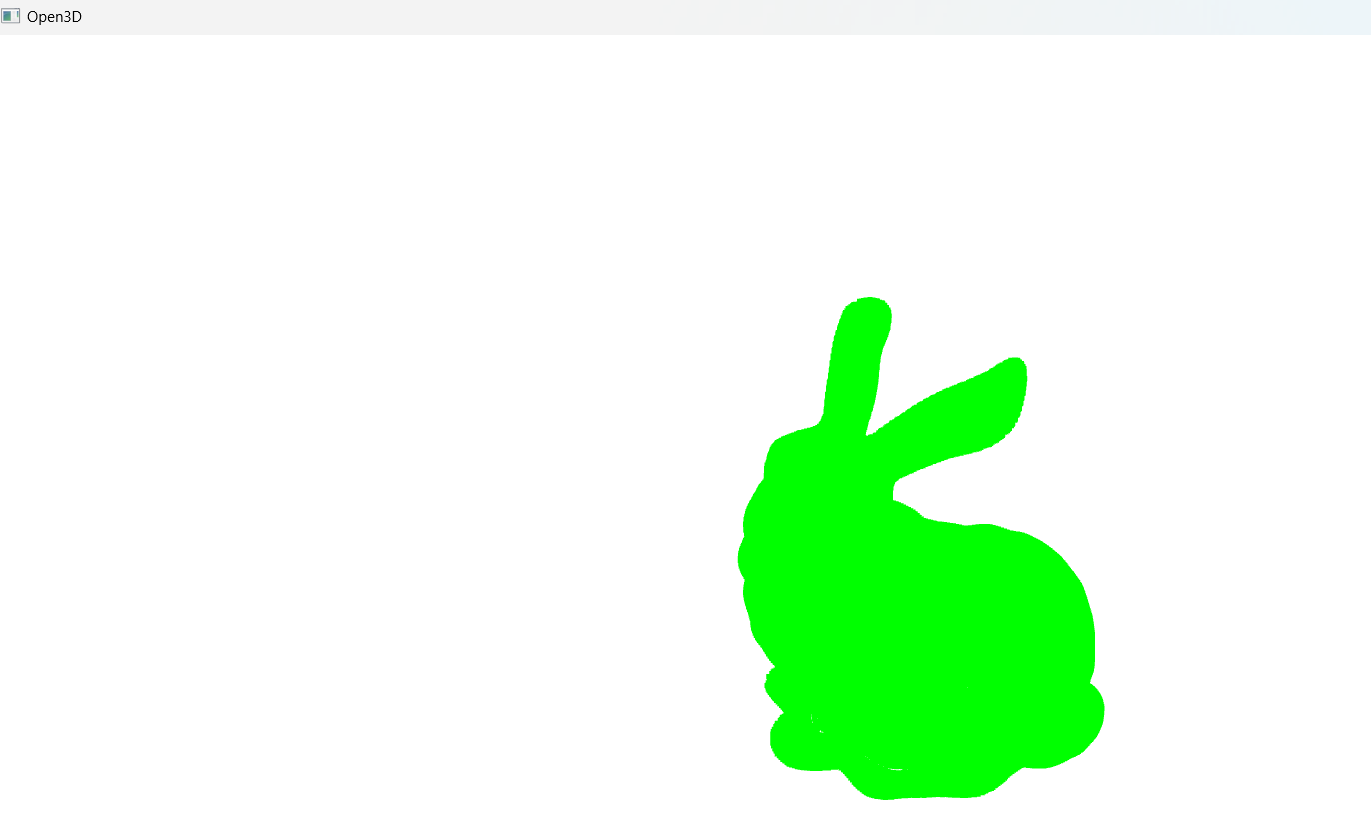
可视化： 读取和加载的点云数据可以通过Open3D的可视化工具进行展示，从而更好地理解数据的结构和内容。

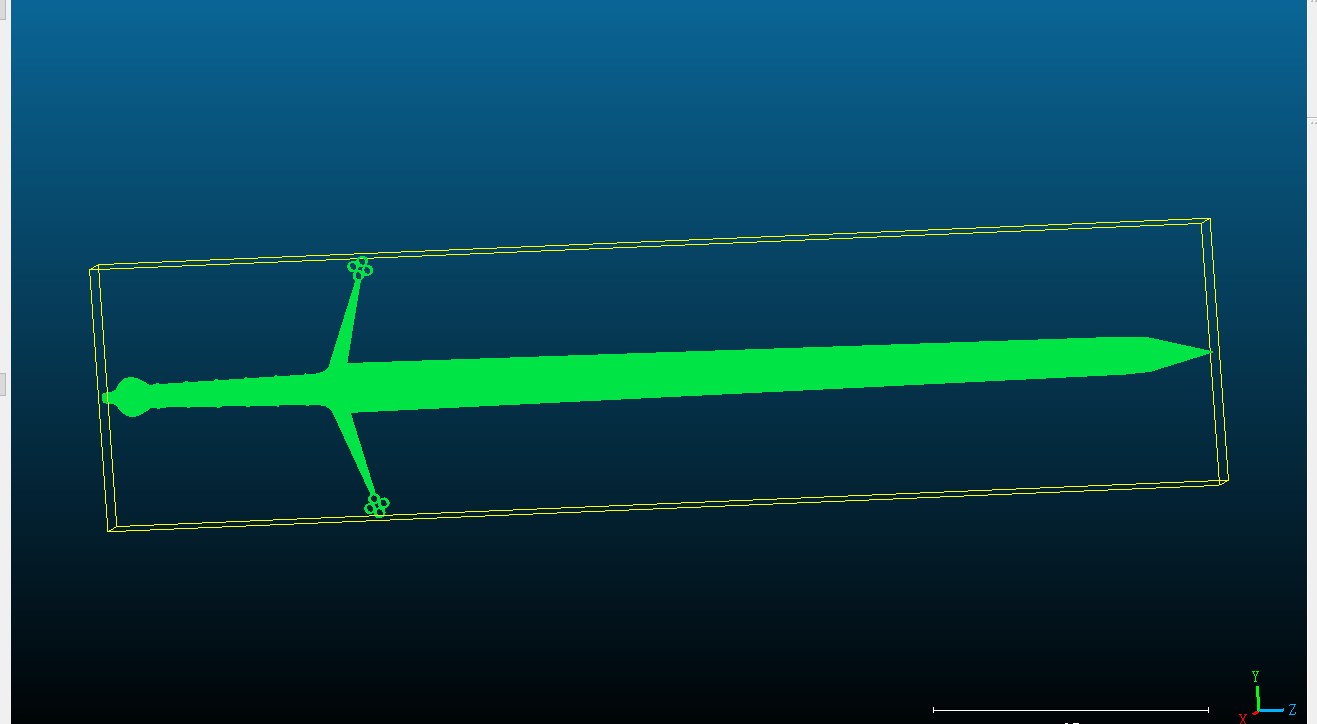
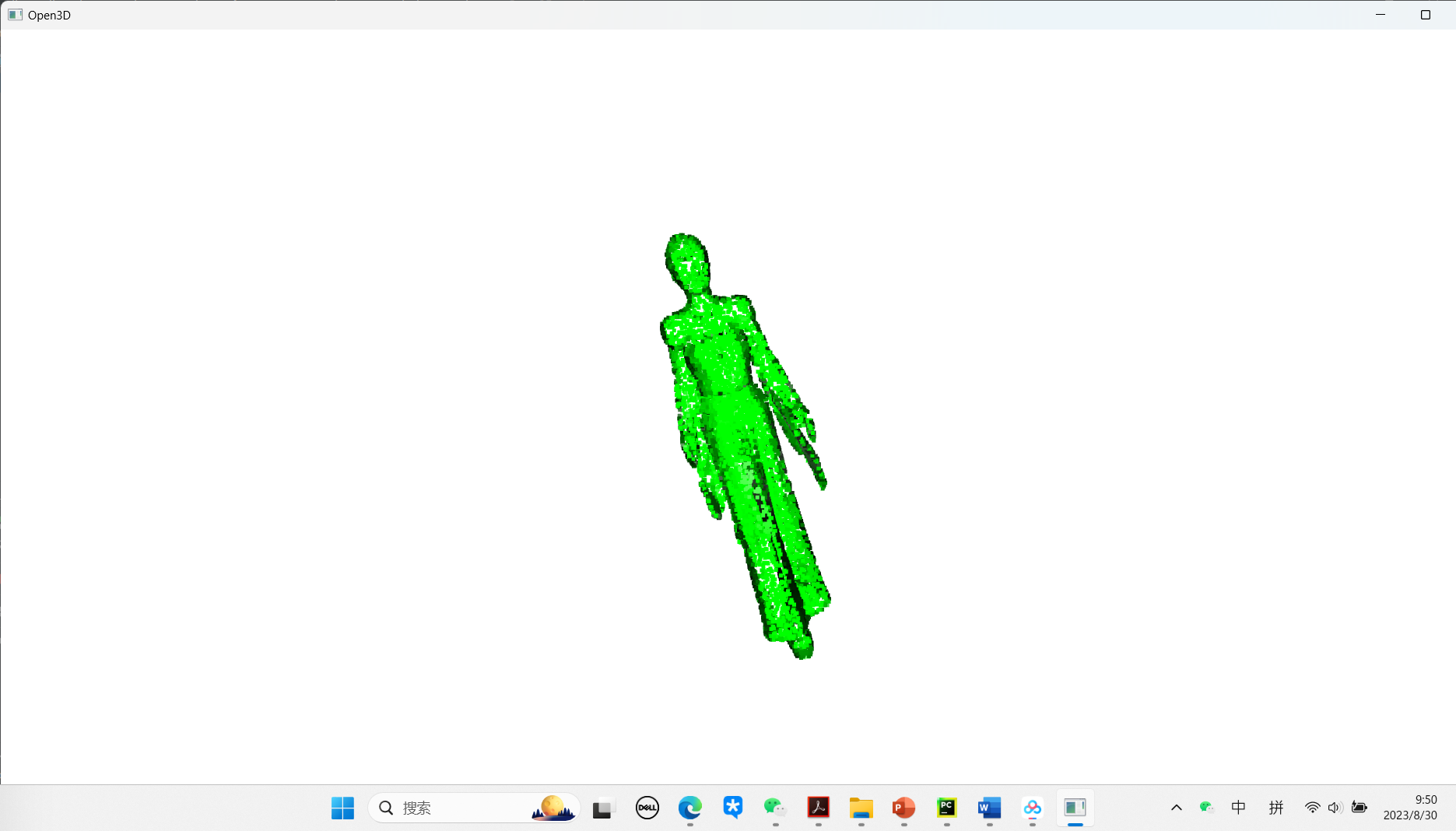
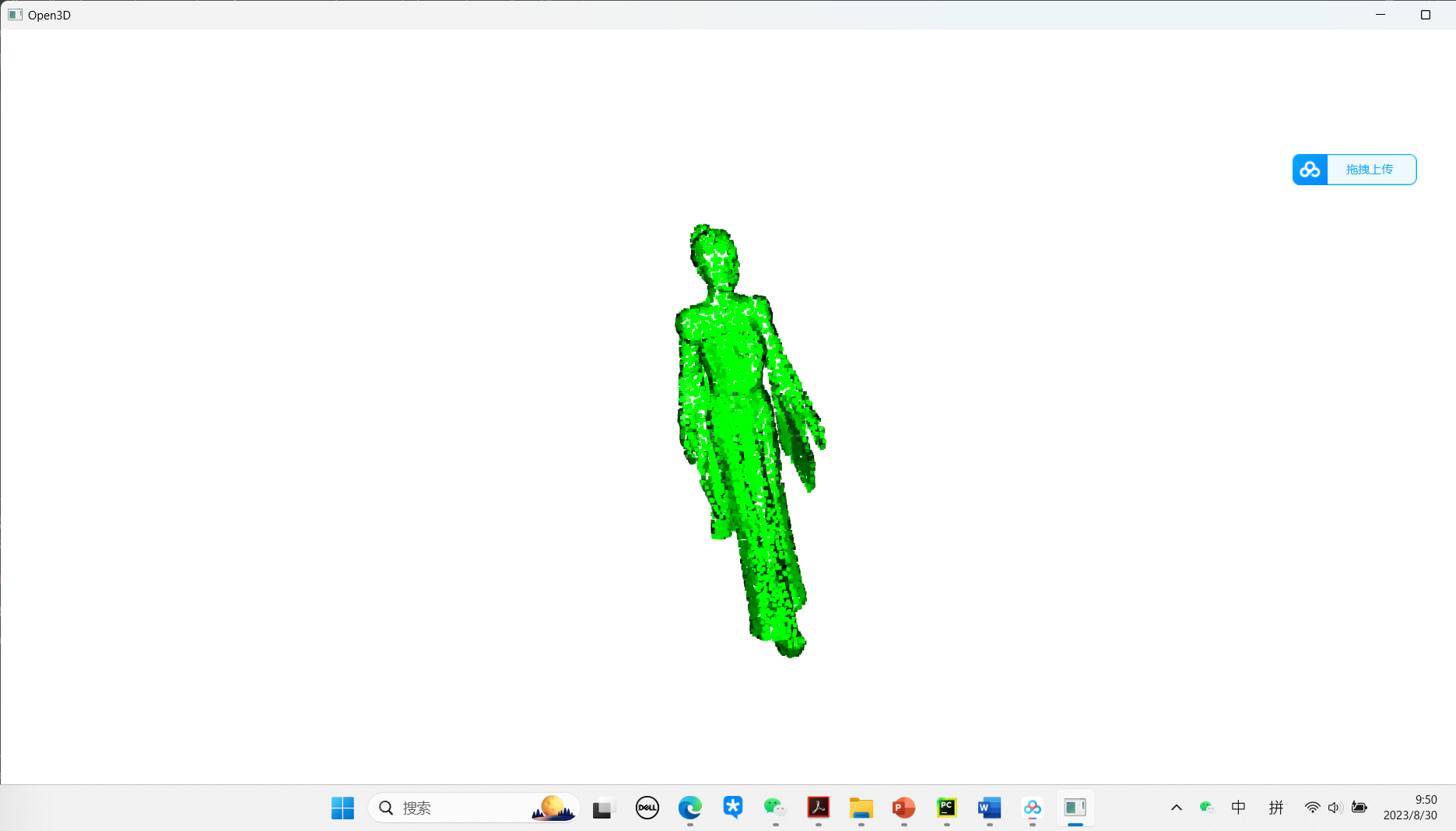
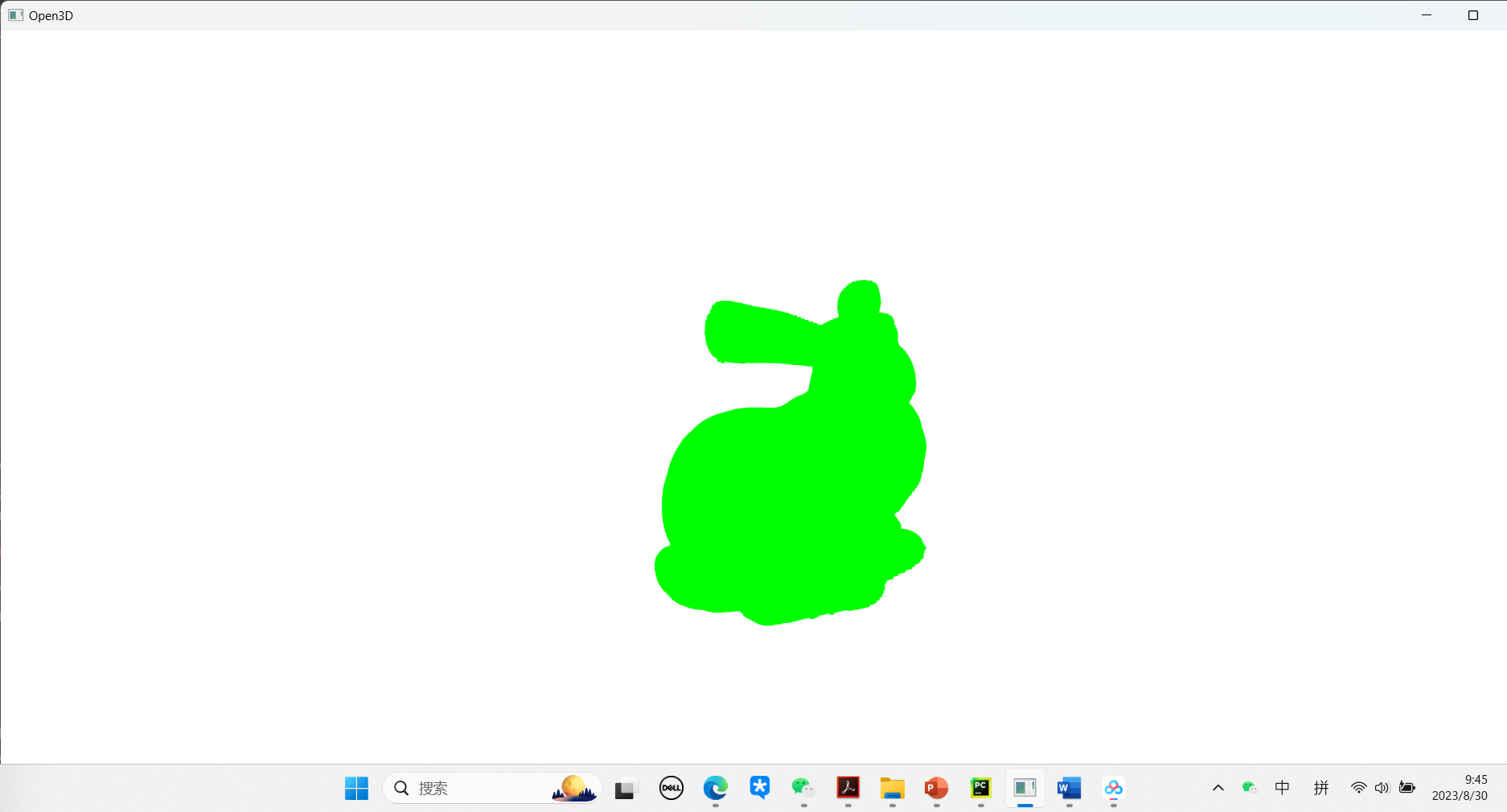




步骤：读取3D效果图

import open3d as o3d  
import numpy as np  
pcd = o3d.io.read\_point\_cloud("person2.pcd")  
print(pcd)#输出点云点的个数  
print(np.asarray(pcd.points))#输出点的三维坐标  
print('给所有的点上一个统一的颜色，颜色是在RGB空间得[0，1]范围内得值')  
pcd.paint\_uniform\_color([0, 1, 0])  
o3d.io.write\_point\_cloud("copy\_of\_fragment.pcd", pcd)  
o3d.visualization.draw\_geometries([pcd])

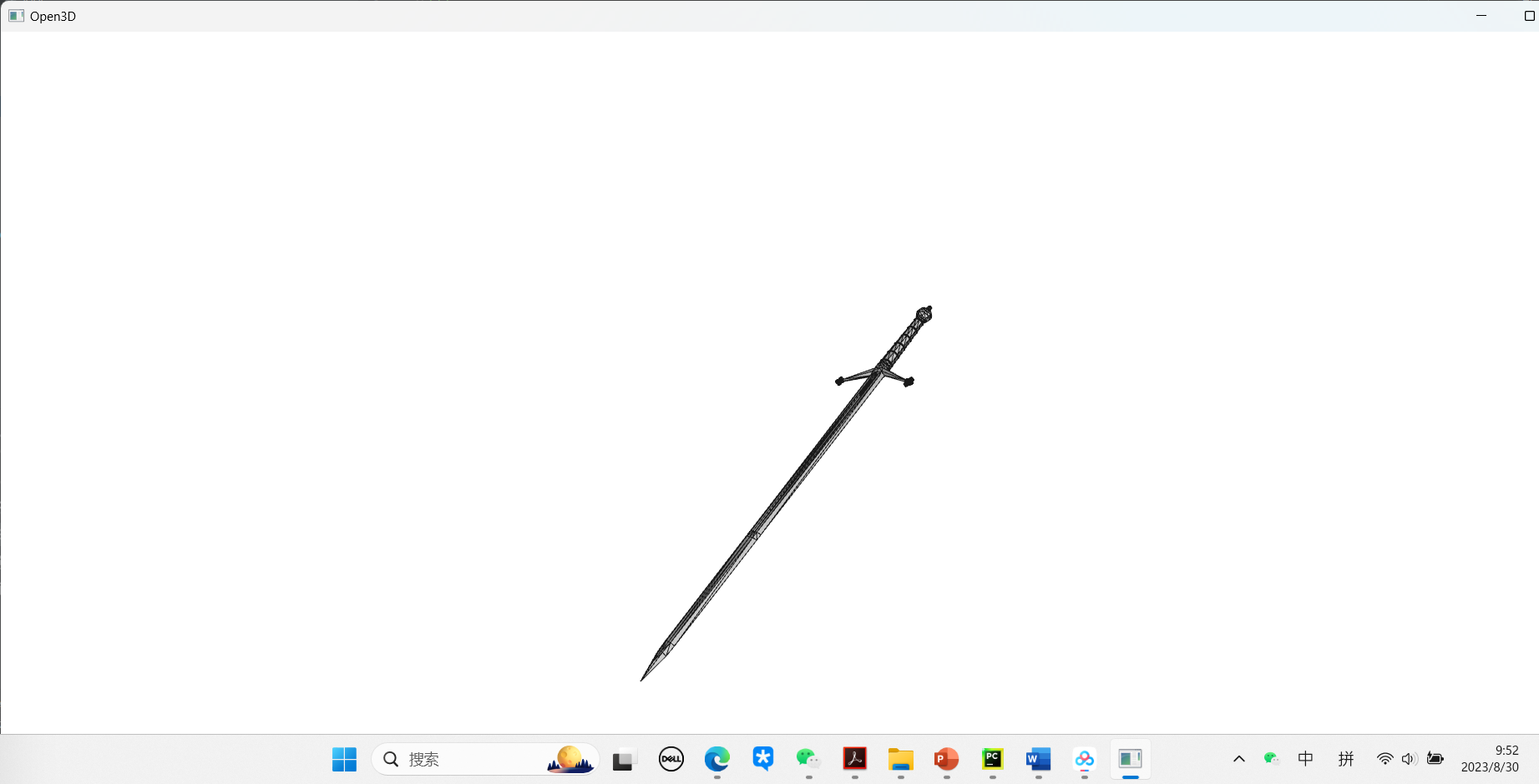


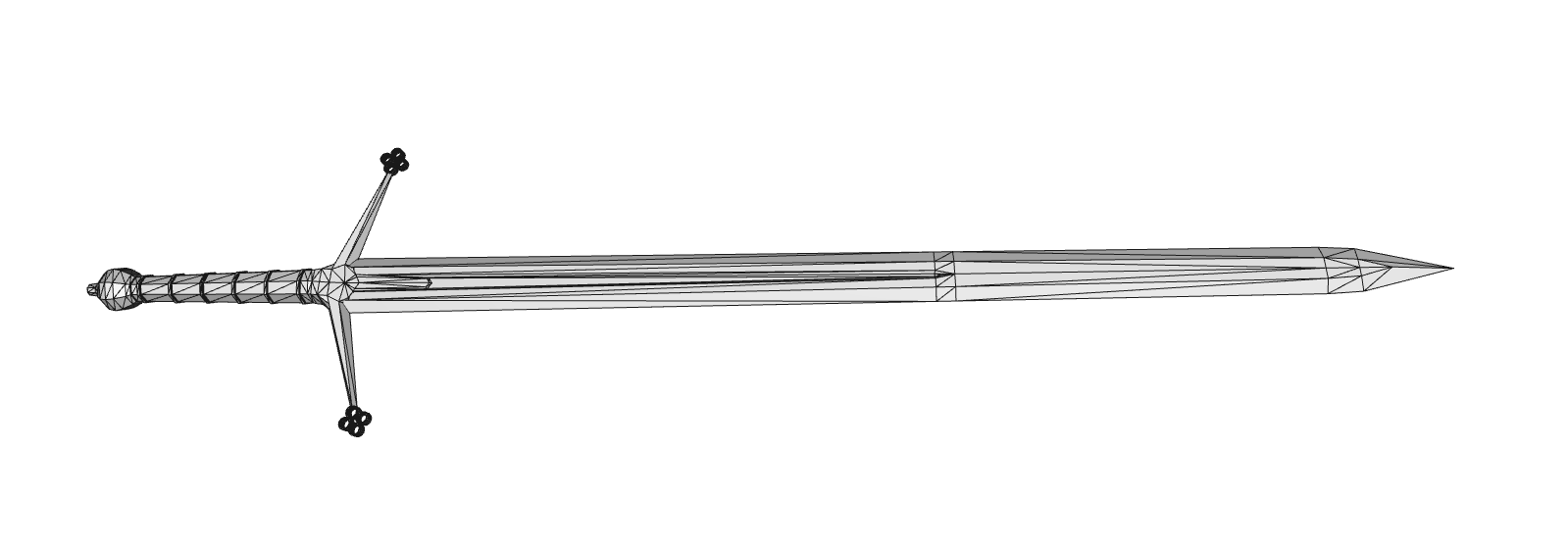


**精细化**

原理：网格精细化处理就是把每个三角形划分为更小的三角形。最简单的方式就是,计算三形每个边的中点,将其划分为四个较小的三角形。这个通过subdivide\_ midpoint 函数实现。3D曲面和面积保持不变,但是顶点和三角形的数量增加了。number\_ \_of\_ iterations 参数定义了精细化处理的次数。

import open3d as o3d  
  
mesh = o3d.io.read\_triangle\_mesh("UV.ply")  
mesh.compute\_vertex\_normals()  
print(  
 f'The mesh has {len(mesh.vertices)} vertices and {len(mesh.triangles)} triangles'  
)  
o3d.visualization.draw\_geometries([mesh], mesh\_show\_wireframe=True)  
mesh = mesh.subdivide\_loop(number\_of\_iterations=2)  
print(  
 f'After subdivision it has {len(mesh.vertices)} vertices and {len(mesh.triangles)} triangles'  
)  
o3d.visualization.draw\_geometries([mesh], mesh\_show\_wireframe=True)

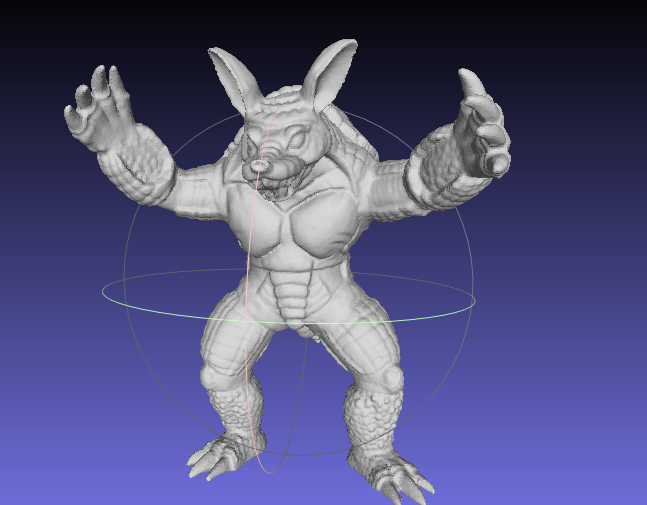




**平滑化**

原理：Open3d包含许多网格滤波的算法，最简单的是均值滤波器,该滤波器能用以网格去噪。一个顶点Vi的值是通过相邻顶点的平均值给出的。公式如下: 

原始数据

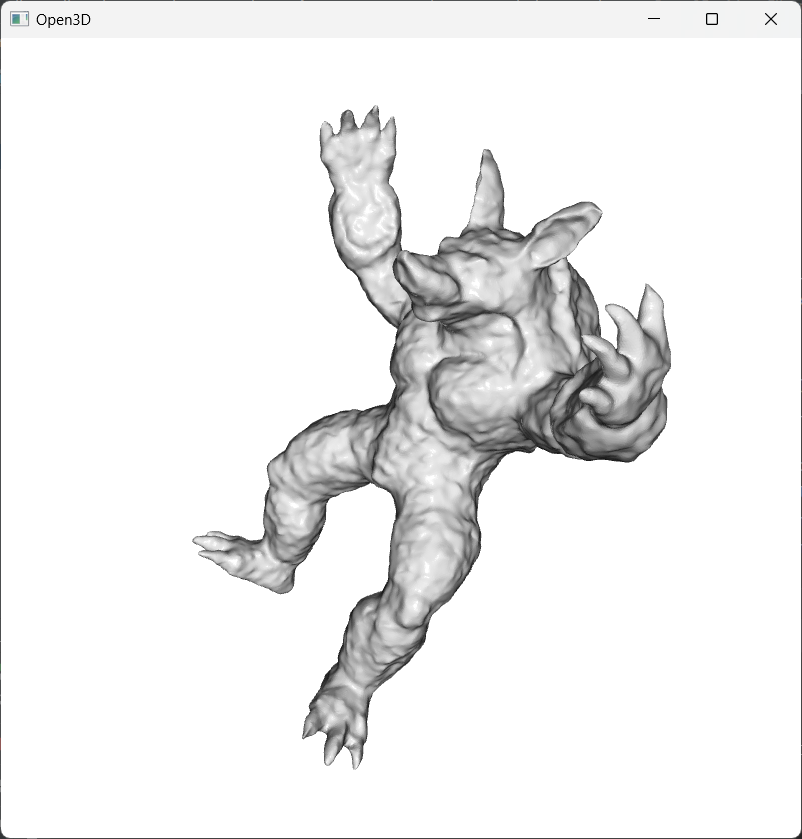


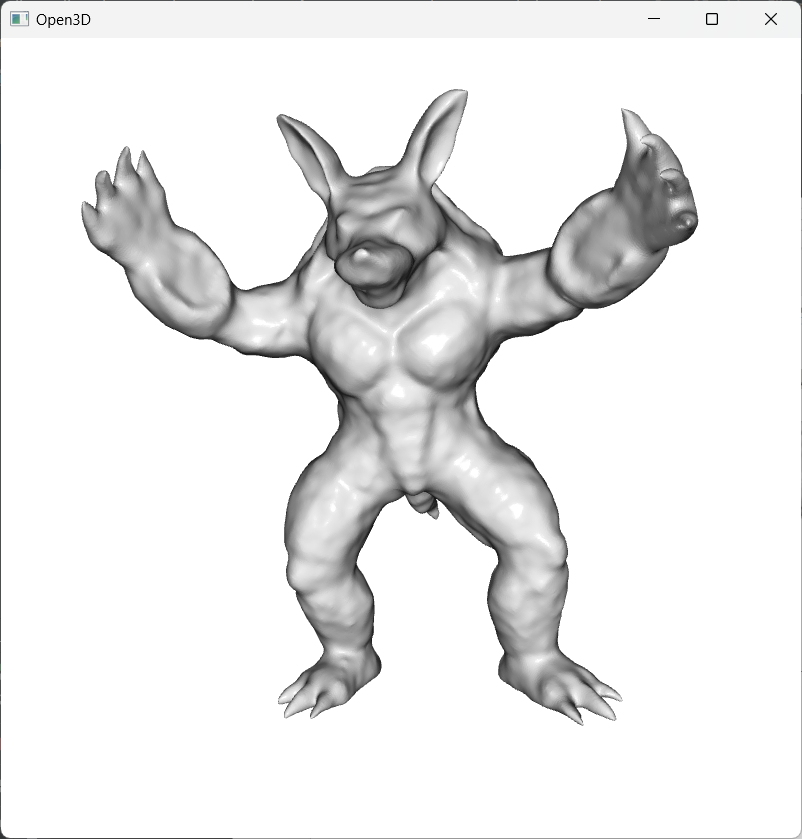
平滑化

经过5次滤波

import open3d as o3d  
import numpy as np  
  
class o3dtut:  
 def get\_knot\_mesh():  
 mesh = o3d.io.read\_triangle\_mesh("Armadillo.ply")  
 mesh.compute\_vertex\_normals()  
 return mesh  
  
mesh\_in = o3dtut.get\_knot\_mesh()  
vertices = np.asarray(mesh\_in.vertices)  
noise = 5  
vertices += np.random.uniform(0, noise, size=vertices.shape)  
mesh\_in.vertices = o3d.utility.Vector3dVector(vertices)  
mesh\_in.compute\_vertex\_normals()  
o3d.visualization.draw\_geometries([mesh\_in],width=800,height=800)  
  
print('filter with average with 1 iteration')  
mesh\_out = mesh\_in.filter\_smooth\_simple(number\_of\_iterations=10)  
mesh\_out.compute\_vertex\_normals()  
o3d.visualization.draw\_geometries([mesh\_out],width=800,height=800)  
  
print('filter with average with 5 iterations')  
mesh\_out = mesh\_in.filter\_smooth\_simple(number\_of\_iterations=20)  
mesh\_out.compute\_vertex\_normals()  
o3d.visualization.draw\_geometries([mesh\_out],width=800,height=800)



滤波10次

滤波20次

**顶点聚类算法**

原理：有时候我们想用较少的三角形来表示一个高份辨率的网格, 但是低分辨率的网格仍然应该接近高分辨率的网格。为此Open3d实现了许多网格简化的算法。

聚类方法将落入给定大小的体素的所有顶点聚合为单个顶点。该访法在simplify\_ \_vertex\_ clustering 中实现，参数

voxel\_ size定义了体素网格的大小，contraction 定义了顶点的聚方式。o3d. geometry . SimplificationContraction . Average计算一个简单的平均值。原始数据

**代码**

import open3d as o3d

mesh = o3d.io.read\_triangle\_mesh("bunny.ply")

mesh.compute\_vertex\_normals()

print(f'Input mesh has {len(mesh.vertices)} vertices and {len(mesh.triangles)} triangles')

o3d.visualization.draw\_geometries([mesh])

voxel\_size = max(mesh.get\_max\_bound() - mesh.get\_min\_bound()) / 32

print(f'voxel\_size = {voxel\_size:e}')

mesh\_smp = mesh.simplify\_vertex\_clustering(

voxel\_size=voxel\_size,

contraction=o3d.geometry.SimplificationContraction.Average)

print(f'Simplified mesh has {len(mesh\_smp.vertices)} vertices and {len(mesh\_smp.triangles)} triangles')

o3d.visualization.draw\_geometries([mesh\_smp])

voxel\_size = max(mesh.get\_max\_bound() - mesh.get\_min\_bound()) / 16

print(f'voxel\_size = {voxel\_size:e}')

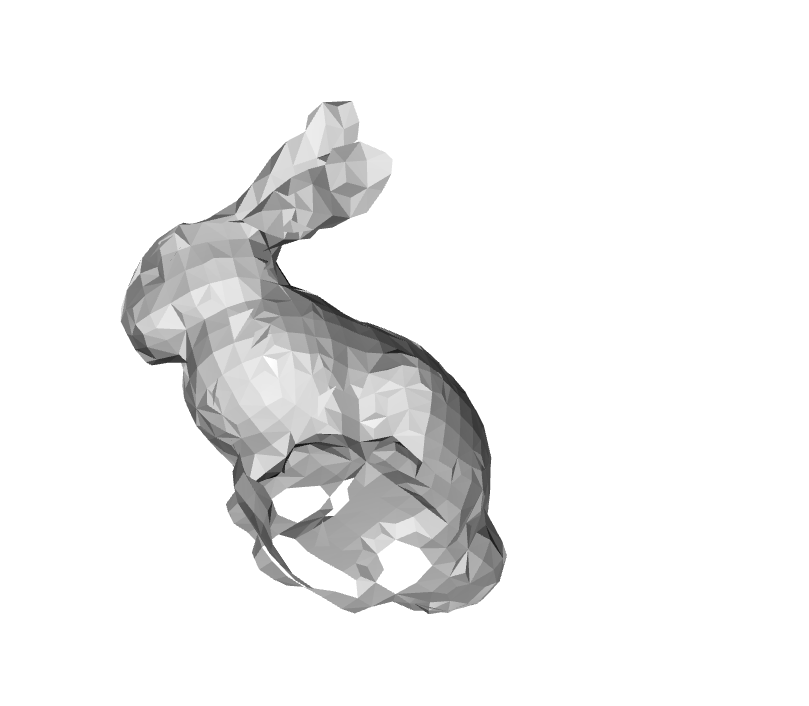
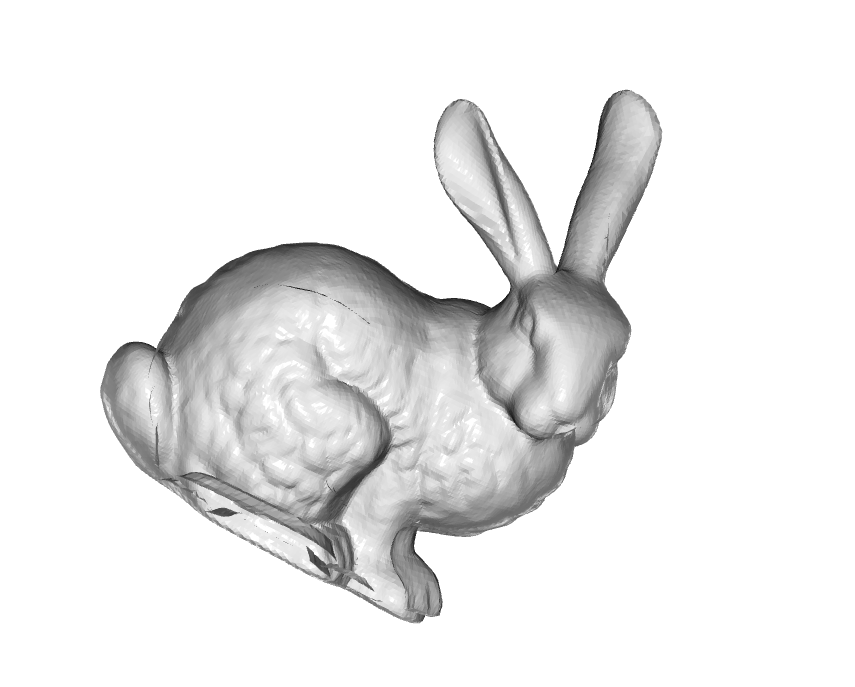
mesh\_smp = mesh.simplify\_vertex\_clustering(

voxel\_size=voxel\_size,

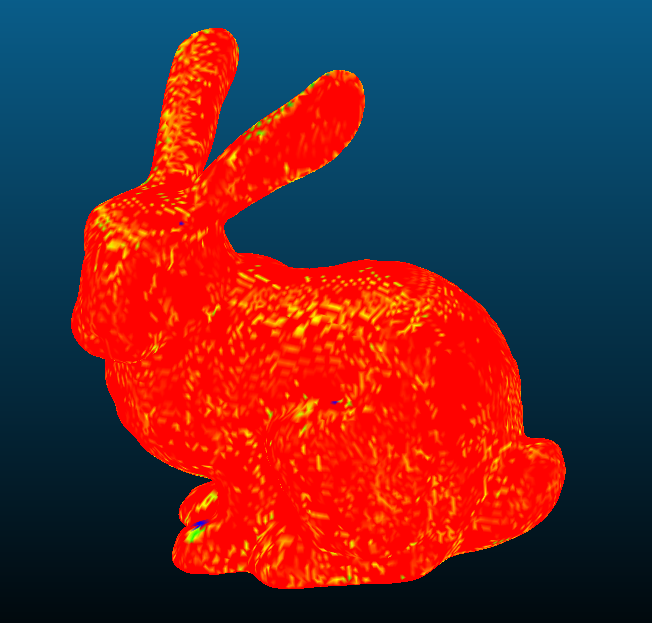
contraction=o3d.geometry.SimplificationContraction.Average)

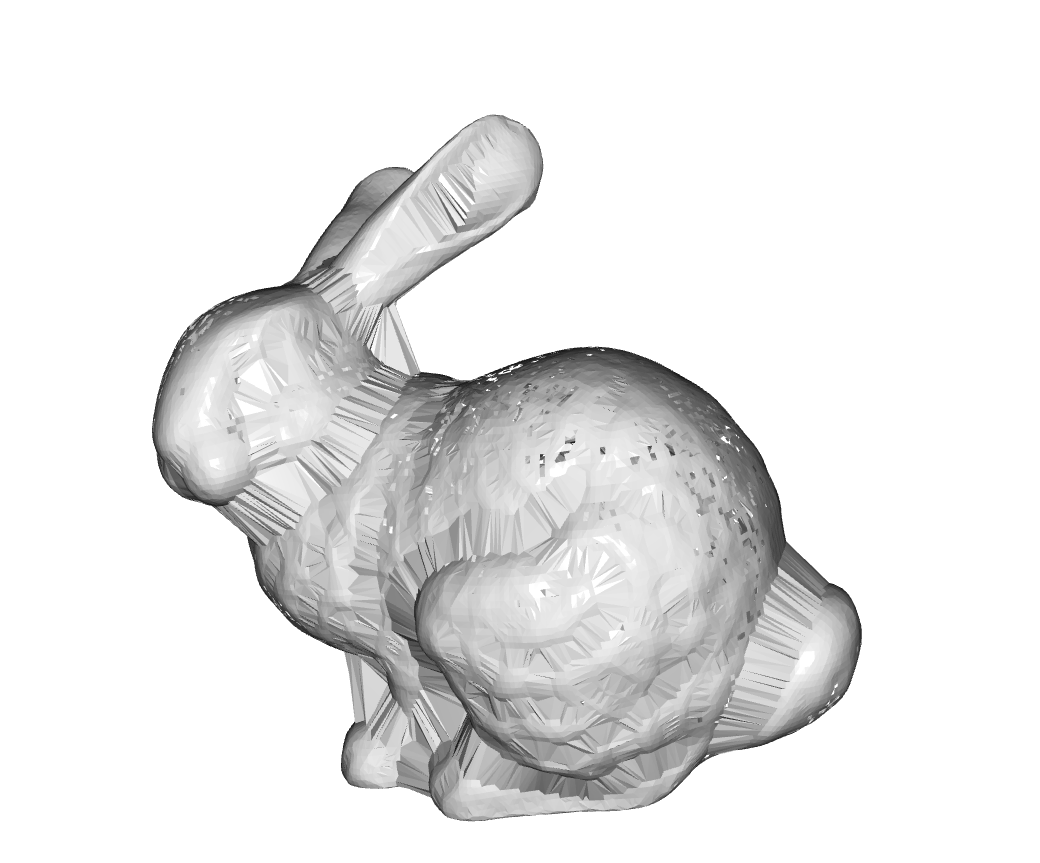
print(f'Simplified mesh has {len(mesh\_smp.vertices)} vertices and {len(mesh\_smp.triangles)} triangles')

o3d.visualization.draw\_geometries([mesh\_smp])



曲面重建





**PointNet++介绍**

PointNet++（PointNet++）是一个用于处理点云数据的深度学习架构，是对原始PointNet模型的扩展和改进。PointNet++ 由Charles R. Qi等人于2017年提出，它在处理三维数据中的点云方面具有很大的优势。与传统的卷积神经网络（CNN）用于处理图像数据类似，PointNet++专门为点云数据设计了有效的神经网络结构。

PointNet++ 的主要目标是解决点云数据的不变性和局部性问题。点云数据在三维空间中由一组点组成，这些点的排列和数量可能是不规则的。因此，传统的卷积操作难以直接应用于点云数据。PointNet++通过层级的方式捕捉点云的局部和全局特征，使得网络能够对点云数据进行更好的理解和处理。

PointNet++ 的架构主要包括以下几个关键思想和模块：

层级结构：PointNet++使用层级的结构来逐步捕捉点云数据的局部特征。它通过一系列的采样和聚合操作，将原始点云数据划分为多个不同层次的局部区域。

邻域采样：为了处理点云数据中的不规则性，PointNet++使用邻域采样操作，从原始点云中选取一组关键点，形成一个新的子集。这样可以有效减少计算量，同时保留了重要的局部信息。

邻域聚合：在每个层级中，PointNet++使用类似于池化操作的方式对邻域内的特征进行聚合，从而捕捉局部特征。这种聚合操作可以在不同层级间逐渐扩展以获取更多的上下文信息。

多尺度处理：PointNet++在不同的层级使用了不同的采样密度，这使得网络能够处理不同尺度的特征。这对于点云数据中的物体检测、分割等任务非常有帮助。

总体而言，PointNet++通过设计适应点云数据特点的网络结构，成功地解决了点云数据处理的挑战。它在诸如语义分割、目标识别、场景分类等任务中取得了显著的成果，成为了处理三维数据的重要工具之一。

PointNet++ 应用介绍

PointNet++ 在处理点云数据方面表现出色，因此在许多领域中都得到了应用。以下是一些 PointNet++ 可能应用的领域：

计算机视觉： PointNet++ 在三维物体检测、分割和识别等任务中具有潜力。它可以用于处理从激光雷达、深度摄像头等传感器获取的三维点云数据，实现更精确的物体识别和检测。

自动驾驶： 在自动驾驶领域，三维点云数据对于环境感知和障碍物检测至关重要。PointNet++ 可以用于处理激光雷达或摄像头获取的点云数据，帮助车辆感知周围环境并做出安全的驾驶决策。

机器人学： 机器人感知和导航需要对周围环境进行精确的建模。PointNet++ 可以用于处理机器人获取的点云数据，实现环境建模、路径规划和避障等功能。

医疗图像处理： 在医学图像处理中，如医学影像的分割和分析，PointNet++ 可以用于处理三维医学图像数据，帮助医生进行疾病诊断和治疗规划。

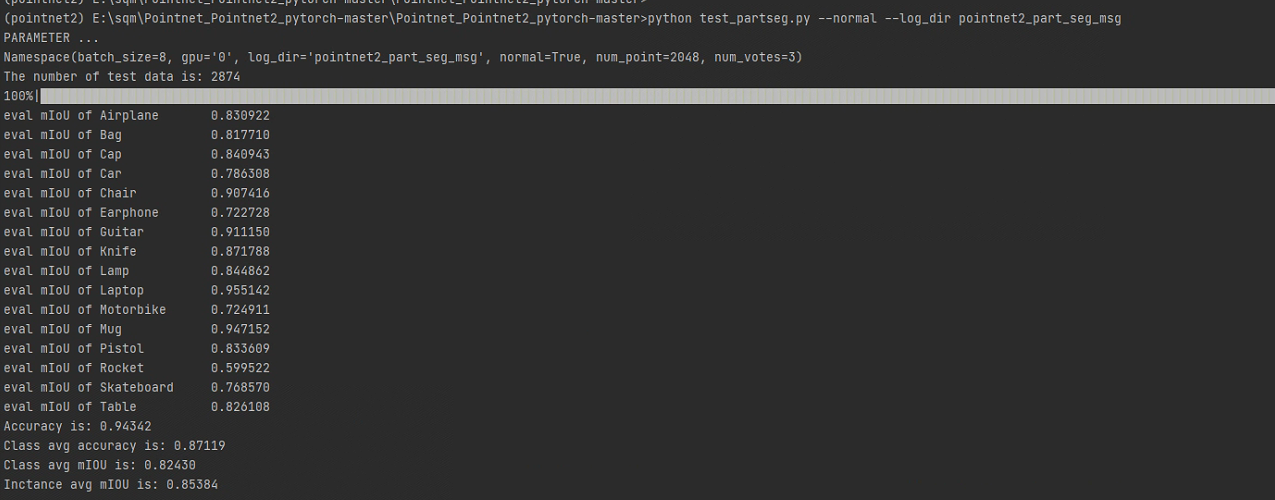
增强现实和虚拟现实： 在 AR 和 VR 应用中，PointNet++ 可以用于将现实世界的三维场景与虚拟元素进行融合，创造更逼真的虚拟体验。

建筑与工程： PointNet++ 可以用于处理建筑物的三维扫描数据，进行建筑物分析、结构检测和维护等。

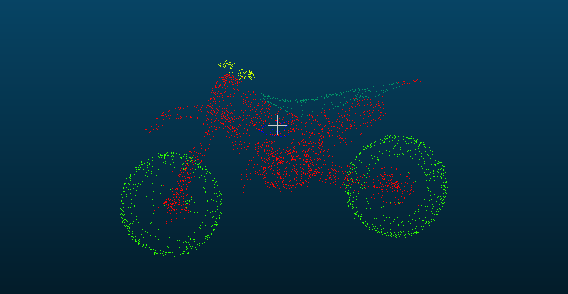
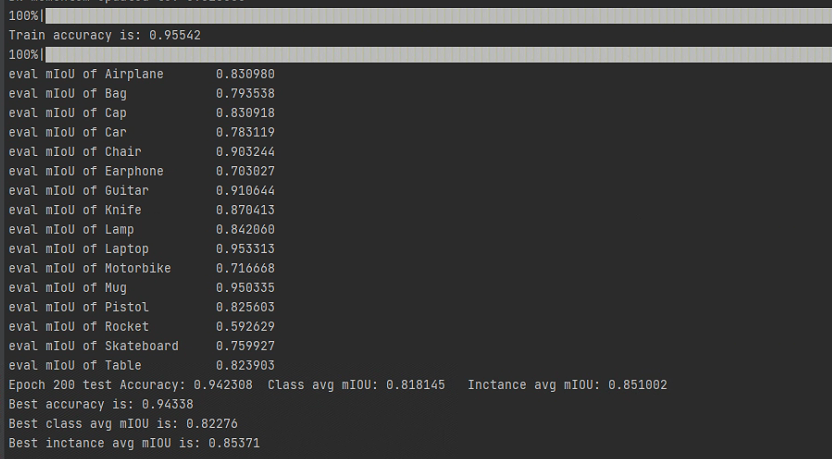
环境监测与地理信息系统： 在环境监测和地理信息系统中，PointNet++ 可以用于处理地形数据、城市模型等三维地理信息，用于灾害预测、规划和资源管理。

总之，PointNet++ 在许多领域中都有潜在的应用，特别是涉及到处理点云数据的任务。它的层级结构和对点云数据的理解能力使其成为处理三维信息的有力工具，有助于提高任务的准确性和效率。

点云分类



点云分割



相关代码的关联性:比如说 先用分割，再去平滑，

先用精细化，再分割

以及一些算法的背景，什么场景用什么。