# 数字媒体处理技术实验报告——实验（四）

### 21R0361班 2021113117-王宇轩

## 目录



## 1. 实验环境

- 操作系统：Windows
- 编程语言：Python
- 所用软件：PyCharm
- 主要库调用：

| 名称 | 介绍 |
| --- | --- |
| open3d | 三维数据处理和几何计算库，本次实验中用于深度信息转点云 |
| pyvista | 三维数据可视化和分析的库，本次实验中用于点云转Mesh网格及可视化 |
| OpenCV | 本次实验中用于图像处理和SGBM对象生成 |

## 2. 文件列表

| 文件名 | 内容 |
| --- | --- |
| stereoConfig.py | 镜头信息文件 |
| stereo.py | 双目图像获取深度信息、生成点云源程序 |
| mesh.py | 点云转Mesh源程序 |

| 文件名 | 内容 |
| --- | --- |
| anothermesh.py | 仅用open3d和numpy库完成的转Mesh源程序 |
| bunny.ply | 用于转Mesh的点云信息输入文件 |
| 实验报告.pdf | 实验报告 |

*已经当堂验收过了，所以此次没有附上验收视频。*双目图像数据选用的是MiddleBury2014数据集中的Backpack-Perfect，由于文件较大不便一起打包，可于https://vision.middlebury.edu/stereo/data/scenes2014/zip/Backpack-perfect.zip下载，在stereo.py同级目录解压到压缩包同名文件夹即可使用。

# 3. 实验内容

## 3.1 双目图像获取深度信息

从双目图像获取深度信息的大致流程如下：双目标定、立体校正（含消除畸变）、立体匹配、视差计算、深度计算(3D坐标)。

对于相机畸变，OpenCV库提供了用于消除畸变的函数：其中参数camera_matrix为相机内参矩阵，存放于文件stereoConfig.py，我们调用的是其中对应MiddleBurry数据集的`setMiddleBurryParams()`。

```
undistortion_image = cv2.undistort(image, camera_matrix, dist_coeff)
```

双目标定的目标是获得左右两个相机的内参、外参和畸变系数，其中内参包括左右相机的fx，fy，cx，cy，外参包括左相机相对于右相机的旋转矩阵和平移向量，畸变系数包括径向畸变系数（k1， k2，k3）和切向畸变系数（p1，p2）以及其他一些畸变类型，都存放于文件stereoConfig.py。

对于立体校正，可调用OpenCV中实现的`stereoRectify()`函数，内部采用的是Bouguet的极线校正算法，算该法需要左右相机的外参[R|T]作为输入。

```
# 获取畸变校正和立体校正的映射变换矩阵、重投影矩阵
# @param: config是一个类，存储着双目标定的参数:config = stereoconfig.stereoCamera()
def getRectifyTransform(height, width, config):
    # 读取内参和外参
    left_K = config.cam_matrix_left
    right_K = config.cam_matrix_right
    left_distortion = config.distortion_l
    right_distortion = config.distortion_r
    R = config.R
    T = config.T

    # 计算校正变换
```

```
    R1, R2, P1, P2, Q, roi1, roi2 = cv2.stereoRectify(left_K, left_distortion, right_K,
right_distortion,
                                                      (width, height), R, T, alpha=0)

    map1x, map1y = cv2.initUndistortRectifyMap(left_K, left_distortion, R1, P1, (width,
height), cv2.CV_32FC1)
    map2x, map2y = cv2.initUndistortRectifyMap(right_K, right_distortion, R2, P2,
(width, height), cv2.CV_32FC1)

    return map1x, map1y, map2x, map2y, Q


# 畸变校正和立体校正
def rectifyImage(image1, image2, map1x, map1y, map2x, map2y):
    rectifyed_img1 = cv2.remap(image1, map1x, map1y, cv2.INTER_AREA)
    rectifyed_img2 = cv2.remap(image2, map2x, map2y, cv2.INTER_AREA)

    return rectifyed_img1, rectifyed_img2


# 立体校正检验----画线
def draw_line(image1, image2):
    # 建立输出图像
    height = max(image1.shape[0], image2.shape[0])
    width = image1.shape[1] + image2.shape[1]

    output = np.zeros((height, width, 3), dtype=np.uint8)
    output[0:image1.shape[0], 0:image1.shape[1]] = image1
    output[0:image2.shape[0], image1.shape[1]:] = image2

    # 绘制等间距平行线
    line_interval = 50  # 直线间隔：50
    for k in range(height // line_interval):
        cv2.line(output, (0, line_interval * (k + 1)), (2 * width, line_interval * (k +
1)), (0, 255, 0), thickness=2,
                 lineType=cv2.LINE_AA)

    return output
```

立体匹配的目的是为左图中的每一个像素点在右图中找到其对应点（世界中相同的物理点），这样就可以计算出视差：$disparity = u_l - u_r$本次实验使用的是OpenCV中实现的SGBM算法。

```
# 视差计算
def stereoMatchSGBM(left_image, right_image, down_scale=False):
    # SGBM匹配参数设置
    if left_image.ndim == 2:
        img_channels = 1
    else:
        img_channels = 3
    blockSize = 3
    paraml = {'minDisparity': 0,
```

```
            'numDisparities': 128,
            'blockSize': blockSize,
            'P1': 8 * img_channels * blockSize ** 2,
            'P2': 32 * img_channels * blockSize ** 2,
            'disp12MaxDiff': 1,
            'preFilterCap': 63,
            'uniquenessRatio': 15,
            'speckleWindowSize': 100,
            'speckleRange': 1,
            'mode': cv2.STEREO_SGBM_MODE_SGBM_3WAY
            }

    # 构建SGBM对象
    left_matcher = cv2.StereoSGBM_create(**paraml)
    paramr = paraml
    paramr['minDisparity'] = -paraml['numDisparities']
    right_matcher = cv2.StereoSGBM_create(**paramr)

    # 计算视差图
    size = (left_image.shape[1], left_image.shape[0])
    if down_scale == False:
        disparity_left = left_matcher.compute(left_image, right_image)
        disparity_right = right_matcher.compute(right_image, left_image)
    else:
        left_image_down = cv2.pyrDown(left_image)
        right_image_down = cv2.pyrDown(right_image)
        factor = left_image.shape[1] / left_image_down.shape[1]

        disparity_left_half = left_matcher.compute(left_image_down, right_image_down)
        disparity_right_half = right_matcher.compute(right_image_down, left_image_down)
        disparity_left = cv2.resize(disparity_left_half, size,
interpolation=cv2.INTER_AREA)
        disparity_right = cv2.resize(disparity_right_half, size,
interpolation=cv2.INTER_AREA)
        disparity_left = factor * disparity_left
        disparity_right = factor * disparity_right

    # 真实视差（因为SGBM算法得到的视差是×16的）
    trueDisp_left = disparity_left.astype(np.float32) / 16.
    trueDisp_right = disparity_right.astype(np.float32) / 16.

    return trueDisp_left, trueDisp_right
```

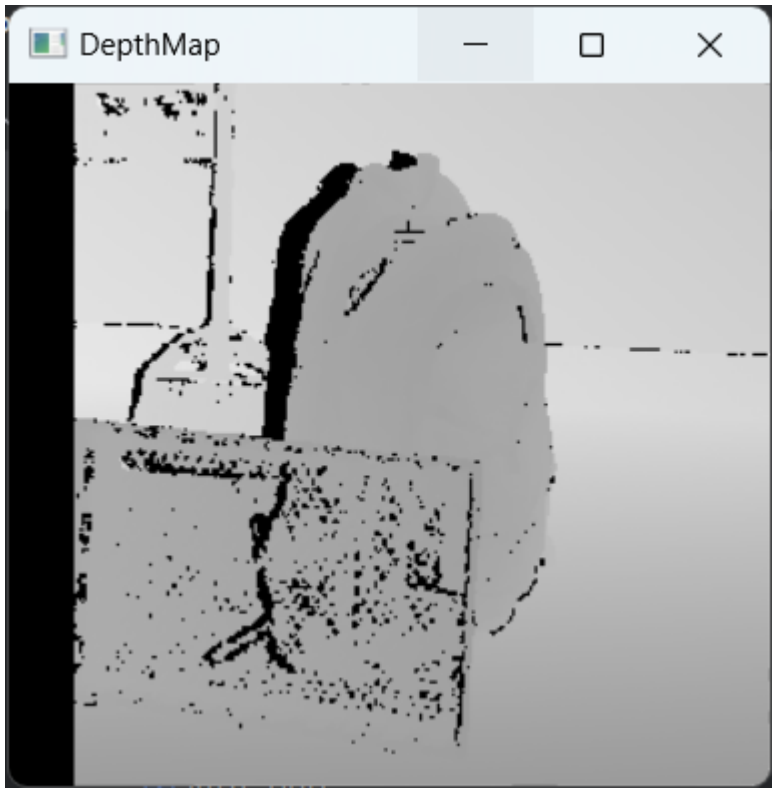
由视差构建深度图：

$$depth = \frac{f \times b}{d + (c_{xr} - c_{xl})}$$

在opencv中使用StereoRectify()函数可以得到一个重投影矩阵Q，使用Q矩阵也可以将像素坐标转换为三维坐标。

```
def getDepthMapWithQ(disparityMap: np.ndarray, Q: np.ndarray) -> np.ndarray:
    points_3d = cv2.reprojectImageTo3D(disparityMap, Q)
    depthMap = points_3d[:, :, 2]
    reset_index = np.where(np.logical_or(depthMap < 0.0, depthMap > 65535.0))
    depthMap[reset_index] = 0

    return depthMap.astype(np.float32)


def getDepthMapWithConfig(disparityMap: np.ndarray, config: stereoConfig.stereoCamera)
-> np.ndarray:
    fb = config.cam_matrix_left[0, 0] * (-config.T[0])
    doffs = config.doffs
    depthMap = np.divide(fb, disparityMap + doffs)
    reset_index = np.where(np.logical_or(depthMap < 0.0, depthMap > 65535.0))
    depthMap[reset_index] = 0
    reset_index2 = np.where(disparityMap < 0.0)
    depthMap[reset_index2] = 0
    return depthMap.astype(np.float32)
```

深度图显示如下。



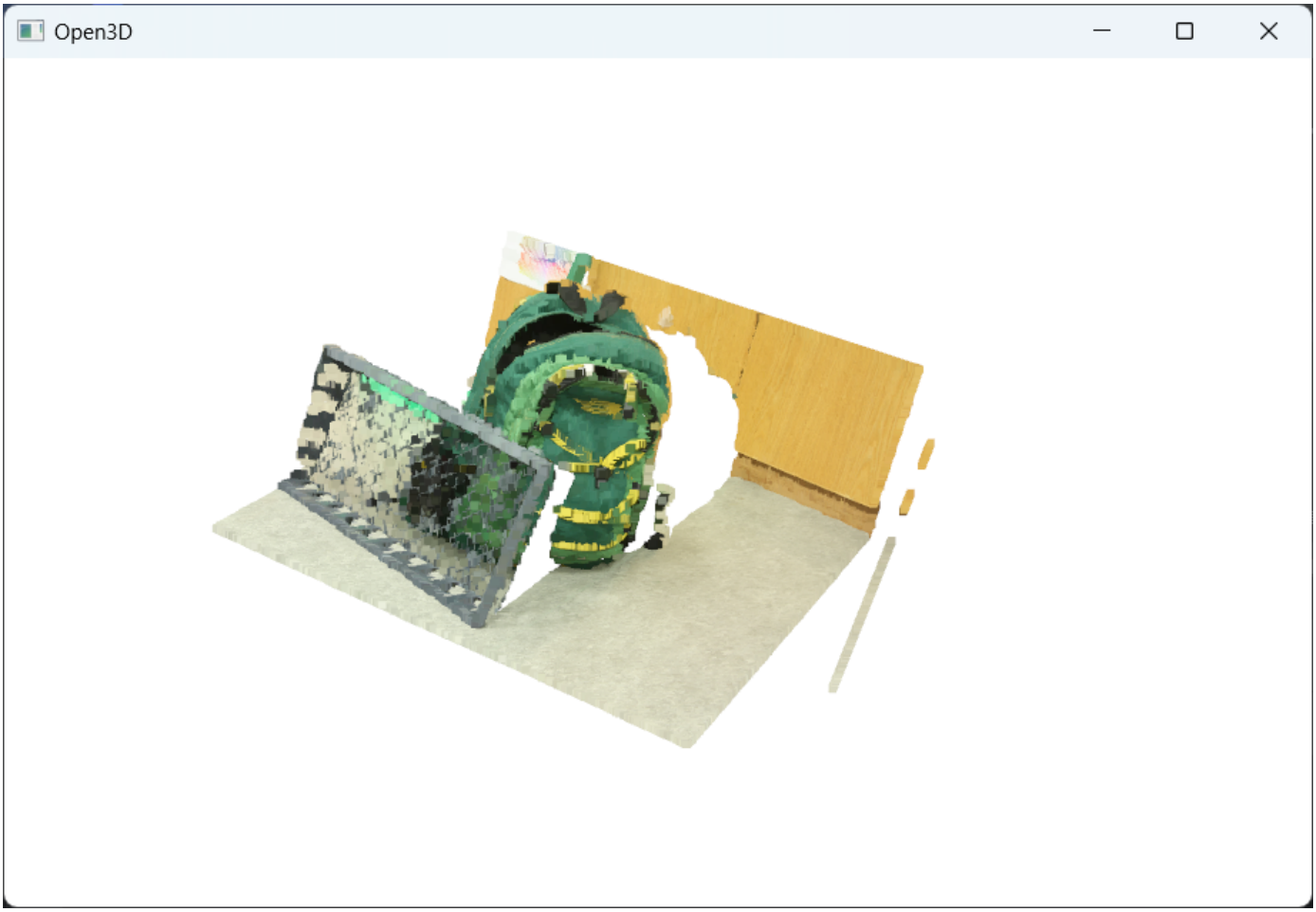

## 3.2 生成点云

使用open3d库绘制点云：

```
    iml = cv2.cvtColor(iml, cv2.COLOR_BGR2RGB)
    colorImage = o3d.geometry.Image(iml)
    depthImage = o3d.geometry.Image(depthMap)
```

```
    rgbdImage = o3d.geometry.RGBDImage.create_from_color_and_depth(color=colorImage,
depth=depthImage,
                                                                   depth_scale=1000.0,
                                                                   depth_trunc=np.inf,
convert_rgb_to_intensity=False)
    intrinsics = o3d.camera.PinholeCameraIntrinsic()
    fx = config.cam_matrix_left[0, 0]
    fy = fx
    cx = config.cam_matrix_left[0, 2]
    cy = config.cam_matrix_left[1, 2]
    print(fx, fy, cx, cy)
    intrinsics.set_intrinsics(width, height, fx=fx, fy=fy, cx=cx, cy=cy)
    extrinsics = np.array([[1., 0., 0., 0.],
                           [0., 1., 0., 0.],
                           [0., 0., 1., 0.],
                           [0., 0., 0., 1.]])
    pointcloud = o3d.geometry.PointCloud().create_from_rgbd_image(rgbdImage,
intrinsic=intrinsics, extrinsic=extrinsics)

    # 计算像素点的3D坐标（左相机坐标系下）
    points_3d = cv2.reprojectImageTo3D(disp, Q)  # 参数中的Q就是由getRectifyTransform()函
数得到的重投影矩阵

    # 构建点云--Point_XYZRGBA格式
    o3d.io.write_point_cloud("PointCloud.pcd", pointcloud=pointcloud)
    o3d.visualization.draw_geometries([pointcloud], width=720, height=480)
```

生成结果如下：

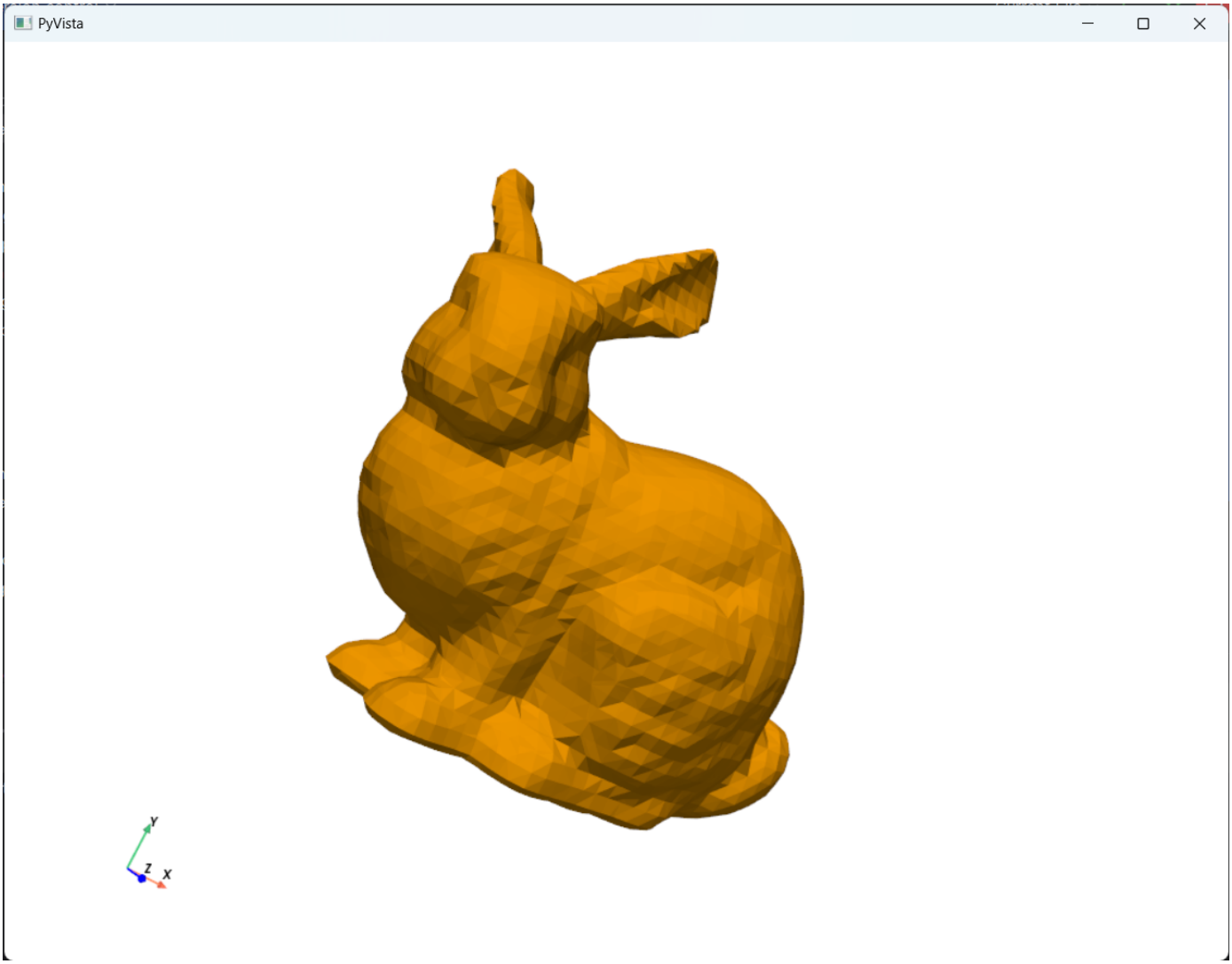



## 3.3 点云转Mesh

PyVista库是一个用于科学可视化和网格处理的Python库，它在内部使用了VTK（Visualization Toolkit）库。将曲面看作一个符号距离函数的等值面，曲面内外的距离值的符号相反，而零等值面即为所求的曲面。该方法需要对点云数据进行网格划分，然后估算每个点的切平面和方向，并以每个点与最近的切平面距离来近似表面距离。这样即可得到一个符号距离的体数据，使用vtkContourFilter来提取零等值面即可得到相应的网格。

```
import numpy as np
import open3d as o3d
import pyvista as pv

# NumPy array with shape (n_points, 3)
# Load saved point cloud
pcd_load = o3d.io.read_point_cloud("bunny.ply")

# convert Open3D.o3d.geometry.PointCloud to numpy array
xyz_load = np.asarray(pcd_load.points)

point_cloud = pv.PolyData(xyz_load)

mesh = point_cloud.reconstruct_surface()

mesh.save('mesh.stl')
mesh.plot(color='orange')
```

生成结果如下：



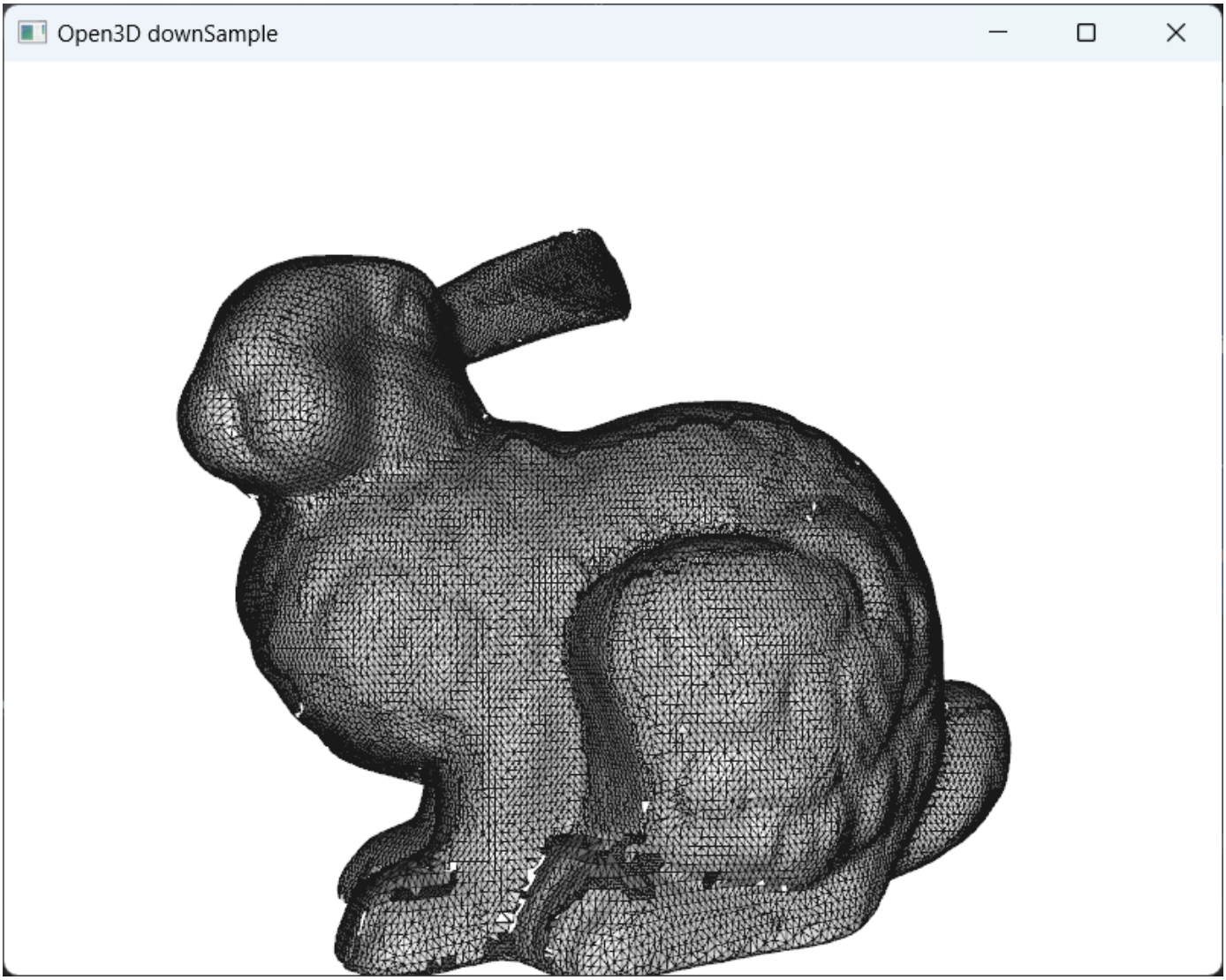

## 3.4 另一种点云转Mesh

尝试了使用open3d中提供的方法来生成mesh。此处使用球旋转算法（Ball-Pivoting Algorithm，BPA），背后的想法是模拟使用虚拟球从点云生成网格，在点云"表面"上滚动一个小球。这个小球取决于网格的比例，并且应该比点之间的平均间距略大。当你将球放到点的表面上时，球会被抓住并落在三个点上，这三个点将形成种子三角形。从那个位置，球沿着由两点形成的三角形边缘滚动。然后球会在一个新的位置安顿下来：一个新的三角形由之前的两个顶点形成，一个新的三角形被添加到网格中。当我们继续滚动和旋转球时，会形成新的三角形并将其添加到网格中。球继续滚动滚动，直到网格完全形成。

```
# 球旋转算法
pc = o3d.io.read_point_cloud("bunny.ply")
pc.estimate_normals()
# 估计球半径
distances = pc.compute_nearest_neighbor_distance()
avg_dist = np.mean(distances)
radius = 1.5 * avg_dist
mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_ball_pivoting(pc,
o3d.utility.DoubleVector([radius, radius * 2]))
print(mesh.get_surface_area())   # 表面积
```

```
o3d.visualization.draw_geometries([mesh], window_name='Open3D downSample', width=800,
height=600, left=50, top=50,
                                  point_show_normal=True, mesh_show_wireframe=True,
mesh_show_back_face=True,)
```

生成结果如下：




## 4. 实验总结

了解了三维重建的基本流程和其中的一些重要算法，掌握了一些开源三维计算库的基本使用。