官方文档：http://www.open3d.org/docs/introduction.html

相比于难用的 PCL 迟迟没有好用的 python binding，以及之前使用 mayavi、vtk、threejs 等对点云做可视化，open3d 作为一个功能完备的工具包，出自 Intel Intelligent Systems Lab，主要的特性有：

- Basic 3D data structures，支持 pcd、ply 等多种数据格式，并且有有好的 numpy 接口可以互相调用。
- Basic 3D data processing algorithms
- Scene reconstruction
- Surface alignment
- 3D visualization
- Python binding

并且具有良好的性能。

安装

```
pip install open3d-python
```

教程

Open3D 提供了丰富的 examples，从 basic 到 advanced，涵盖了数据读写、可视化、三维重建、点云匹配、crop 等各种 demo。

```
Python
├── Advanced
│   ├── camera_trajectory.py
│   ├── color_map_optimization.py
│   ├── colored_pointcloud_registration.py
│   ├── customized_visualization.py
│   ├── downsampling_and_trace.py
│   ├── fast_global_registration.py
│   ├── global_registration.py
│   ├── headless_rendering.py
│   ├── interactive_visualization.py
│   ├── load_save_viewpoint.py
│   ├── multiway_registration.py
│   ├── non_blocking_visualization.py
│   ├── pointcloud_outlier_removal.py
│   ├── remove_geometry.py
│   ├── rgbd_integration.py
│   ├── test.json
│   └── trajectory_io.py
```

```
├── Basic
│   ├── file_io.py
│   ├── half_edge_mesh.py
│   ├── icp_registration.py
│   ├── kdtree.py
│   ├── mesh.py
│   ├── mesh_sampling.py
│   ├── mesh_voxelization.py
│   ├── pointcloud.py
│   ├── python_binding.py
│   ├── rgbd_nyu.py
│   ├── rgbd_odometry.py
│   ├── rgbd_redwood.py
│   ├── rgbd_sun.py
│   ├── rgbd_tum.py
│   ├── visualization.py
│   └── working_with_numpy.py
├── Benchmark
│   ├── benchmark_fgr.py
│   ├── benchmark_pre.py
│   ├── benchmark_ransac.py
├── CMakeLists.txt
├── Misc
│   ├── color_image.py
│   ├── evaluate_geometric_feature.py
│   ├── feature.py
│   ├── pose_graph_optimization.py
│   ├── sampling.py
│   └── vector.py
├── ReconstructionSystem
│   ├── config
│   ├── dataset
│   ├── debug
│   ├── initialize_config.py
│   ├── integrate_scene.py
│   ├── make_fragments.py
│   ├── opencv_pose_estimation.py
│   ├── optimize_posegraph.py
│   ├── refine_registration.py
│   ├── register_fragments.py
│   ├── run_system.py
│   ├── scripts
│   └── sensors
└── Utility
    ├── downloader.py
    ├── file.py
    ├── opencv.py
    └── visualization.py
```

1. 文件读写

下表列出了 open3d 支持的数据类型

Format	Description
`xyz`	Each line contains `[x, y, z]`, where `x, y, z` are the 3D coordinates
`xyzn`	Each line contains `[x, y, z, nx, ny, nz]`, where `nx, ny, nz` are the normals
`xyzrgb`	Each line contains `[x, y, z, r, g, b]`, where `r, g, b` are in floats of range `[0, 1]`
`pts`	The first line is an integer representing the number of pointsEach subsequent line contains `[x, y, z, i, r, g, b]`, where `r, g, b` are in `uint8`
`ply`	See Polygon File Format, the `ply` file can contain both point cloud and mesh
`pcd`	See Point Cloud Data

```
from open3d import *
# 读写 point cloud
pcd = read_point_cloud("../../TestData/fragment.pcd")
pcd = read_point_cloud("my_points.txt", format='xyz')
write_point_cloud("copy_of_fragment.pcd", pcd)
# 读写 mesh
mesh = read_triangle_mesh("../../TestData/knot.ply")
write_triangle_mesh("copy_of_knot.ply", mesh)
# 读写 image
img = read_image("../../TestData/lena_color.jpg")
write_image("copy_of_lena_color.jpg", img)
```

2. visualization

```
points =
np.fromfile("/Users/poodarchu/Development/nuScenes/velodyne/fde3c0ece910452c8a
b6a4c877c95445.bin", dtype=np.float32)
points = points.reshape(-1, 5)[:, :3]
pcd = PointCloud()
pcd.points = Vector3dVector(points)
# pcd.paint_uniform_color([1, 0.706, 0.9])
pcd.paint_uniform_color([0.5, 0.5, 0.5])
draw_geometries([pcd])
```

```
# 下采样
# 分为两步,
# 1. Points are bucketed into voxels.
# 2. Each occupied voxel generates exact one point by averaging all points
inside.
downpcd = voxel_down_sample(pcd, voxel_size = 0.05)
draw_geometries([downpcd])

# estimate normals
# estimate_normals computes normal for every point. The function finds
adjacent points and calculate the principal axis of the adjacent points using
covariance analysis.
# Use draw_geometries to visualize the point cloud and press n to see point
normal. Key - and key + can be used to control the length of the normal.
estimate_normals(downpcd, search_param = KDTreeSearchParamHybrid(radius = 0.1,
max_nn = 30))
draw_geometries([downpcd])

# Crop point cloud using a polygon volume
vol = read_selection_polygon_volume("cropped.json")
chair = vol.crop_point_cloud(pcd)
draw_geometries([chair])
```

3. RGB-D data

Open3D has a data structure for images. It supports various functions such as `read_image`, `write_image`, `filter_image` and `draw_geometries`. An Open3D `Image` can be directly converted to/from a numpy array.

An Open3D `RGBDImage` is composed of two images, `RGBDImage.depth` and `RGBDImage.color`.

example：http://www.open3d.org/docs/tutorial/Basic/rgbd_images/sun.html

RGB-D Odometry

An RGBD odometry finds the camera movement between two consecutive RGBD image pairs.

example: http://www.open3d.org/docs/tutorial/Basic/rgbd_odometry.html

4. KDTree for fast fast retrieval of nearest neighbors

```
# examples/Python/Basic/kdtree.py

import numpy as np
from open3d import *
```

```
if __name__ == "__main__":

    print("Testing kdtree in open3d ...")
    print("Load a point cloud and paint it gray.")
    pcd = read_point_cloud("../../TestData/Feature/cloud_bin_0.pcd")
    pcd.paint_uniform_color([0.5, 0.5, 0.5])
    pcd_tree = KDTreeFlann(pcd)

    print("Paint the 1500th point red.")
    pcd.colors[1500] = [1, 0, 0]

    print("Find its 200 nearest neighbors, paint blue.")
    [k, idx, _] = pcd_tree.search_knn_vector_3d(pcd.points[1500], 200)
    np.asarray(pcd.colors)[idx[1:], :] = [0, 0, 1]

    print("Find its neighbors with distance less than 0.2, paint green.")
    [k, idx, _] = pcd_tree.search_radius_vector_3d(pcd.points[1500], 0.2)
    np.asarray(pcd.colors)[idx[1:], :] = [0, 1, 0]

    print("Visualize the point cloud.")
    draw_geometries([pcd])
    print("")
```

5. ICP(Iterative Closest Point) registration algorithm

The input are two point clouds and an initial transformation that roughly aligns the source point cloud to the target point cloud. The output is a refined transformation that tightly aligns the two point clouds.

A helper function `draw_registration_result` visualizes the alignment during the registration process.

Open3D shows point-to-point ICP and the point-to-plane ICP :

```
# examples/Python/Basic/icp_registration.py

from open3d import *
import numpy as np
import copy

def draw_registration_result(source, target, transformation):
    source_temp = copy.deepcopy(source)
    target_temp = copy.deepcopy(target)
    source_temp.paint_uniform_color([1, 0.706, 0])
    target_temp.paint_uniform_color([0, 0.651, 0.929])
    source_temp.transform(transformation)
    draw_geometries([source_temp, target_temp])
```

```
if __name__ == "__main__":
    source = read_point_cloud("../../TestData/ICP/cloud_bin_0.pcd")
    target = read_point_cloud("../../TestData/ICP/cloud_bin_1.pcd")
    threshold = 0.02
    trans_init = np.asarray(
                [[0.862, 0.011, -0.507,  0.5],
                [-0.139, 0.967, -0.215,  0.7],
                [0.487, 0.255,  0.835, -1.4],
                [0.0, 0.0, 0.0, 1.0]])
    draw_registration_result(source, target, trans_init)
    print("Initial alignment")
    evaluation = evaluate_registration(source, target,
            threshold, trans_init)
    print(evaluation)

    print("Apply point-to-point ICP")
    reg_p2p = registration_icp(source, target, threshold, trans_init,
            TransformationEstimationPointToPoint())
    print(reg_p2p)
    print("Transformation is:")
    print(reg_p2p.transformation)
    print("")
    draw_registration_result(source, target, reg_p2p.transformation)

    print("Apply point-to-plane ICP")
    reg_p2l = registration_icp(source, target, threshold, trans_init,
            TransformationEstimationPointToPlane())
    print(reg_p2l)
    print("Transformation is:")
    print(reg_p2l.transformation)
    print("")
    draw_registration_result(source, target, reg_p2l.transformation)
```

5.1 Point-to-Point ICP

In general, the ICP algorithm iterates over two steps:

> 1. Find correspondence set □={(**p**,**q**)}K={(p,q)} from target point cloud **P**P, and source point cloud **Q**Qtransformed with current transformation matrix **T**T.
> 2. Update the transformation **T**T by minimizing an objective function $E(\mathbf{T})$E(T) defined over the correspondence set □K.

Different variants of ICP use different objective functions $E(\mathbf{T})$E(T) [BeslAndMcKay1992] [ChenAndMedioni1992] [Park2017].

We first show a point-to-point ICP algorithm [BeslAndMcKay1992] using an objective

$E(\mathbf{T})=\sum_{(\mathbf{p},\mathbf{q})\in\square}\|\mathbf{p}-\mathbf{T}\mathbf{*}\mathbf{*}\mathbf{q}\mathbf{*}\mathbf{*}\|2$.

5.2 Point-to-plane ICP

he point-to-plane ICP algorithm [ChenAndMedioni1992] uses a different objective function

$E(\mathbf{T})=\sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}}((\mathbf{p}-\mathbf{T}\mathbf{q})\cdot\mathbf{n}_{\mathbf{p}})^2$,E(T)=∑(p,q)∈K((p−Tq)⋅np)2,

where **$\mathbf{n}_{\mathbf{p}}$np is the normal of point $\mathbf{p}$**p. [Rusinkiewicz2001] has shown that the point-to-plane ICP algorithm has a faster convergence speed than the point-to-point ICP algorithm.

6. Working with numpy

From numpy to open3d

```
# Pass xyz to Open3D.PointCloud and visualize
pcd = PointCloud()
pcd.points = Vector3dVector(xyz)
write_point_cloud("../../TestData/sync.ply", pcd)
```

From open3d to numpy

```
# Load saved point cloud and visualize it
pcd_load = read_point_cloud("../../TestData/sync.ply")
draw_geometries([pcd_load])

# convert Open3D.PointCloud to numpy array
xyz_load = np.asarray(pcd_load.points)
print('xyz_load')
print(xyz_load)
```

From numpy to open3d Image

```
# save z_norm as an image (change [0,1] range to [0,255] range with uint8
type)
img = Image((z_norm*255).astype(np.uint8))
write_image("../../TestData/sync.png", img)
draw_geometries([img])
```

7. Jupyter visualization

support: http://www.open3d.org/docs/tutorial/Basic/jupyter.html

example usage

```
import numpy as np
import open3d as o3
from open3d import JVisualizer

pts_path = "examples/TestData/fragment.ply"
fragment = o3.read_point_cloud(pts_path)
visualizer = JVisualizer()
visualizer.add_geometry(fragment)
visualizer.show()
```

8. Advanced topics

http://www.open3d.org/docs/tutorial/Advanced/index.html