

Goal

The main idea behind this study is to evaluate if a library we have developed at the ipb-lab is easy to use for people from our community(robotics).

You will be implementing a 3D Surface Reconstruction pipeline employing TSDF, for any type of sensor data. Sounds complicated, right? Well, the idea is that when using our library, this should not be.

You must time how long it takes you to perform each of the following 4 steps and report back to us the timings. Please determine the timings for the 4 steps separately (eg: Step1=30s; Step2=10min; Step3=3min; Step4=90s).

Exercise

Before starting

To get started you just need:

- A 64-bit GNU/Linux computer with Python3.8 or newer
- Have a stopwatch ready
- You can pick any 3D dataset of your choice as long as you have good pose estimates for this dataset. Once you have picked your favorite dataset, you are ready to go

Start the timer now!

1. Install the library

- First, make sure you have an updated version of **pip**: **\$ pip install -U pip**. We tested with version 21.3.1
- Then run **pip install vdbfusion** on a terminal to install our library.

2. Writing the Dataloader

You have your favorite dataset, so it's time to start coding! For that you have two options:

1. Use an existing dataloader from your own.
2. Define a new **Dataset** class in Python like the one shown below.

The only restriction is that you need to provide point clouds(in global coordinate frame) in the form of a numpy array and the origin of the sensor in this global coordinate frame, also a numpy array. For what follows, we assume you implemented a **Dataset** using the snippet provided here.

NOTE: Please pay attention in providing your point cloud in a global coordinate frame(this means, you need to have applied the pose you have for that sensor frame before feeding it into the mapping pipeline)

```
1 class Dataset:
2     def __init__(self, *args, **kwargs):
3         # Initialize your dataset here ..
4
5     def __len__(self) -> int:
6         return len(self.n_scans)
7
8     def __getitem__(self, idx: int) -> Tuple[np.ndarray, np.ndarray]:
9         # points: np.array (N, 3), in the global coordinate frame.
10        # origin: np.array(3,), sensor origin in the global coordinate frame.
11        return points, origin
```

3. Writing the fusion pipeline

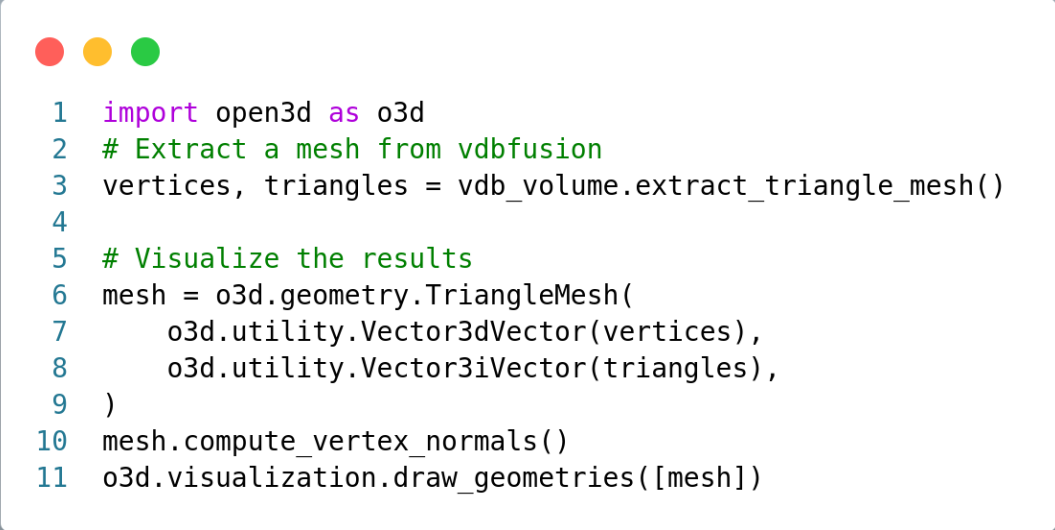
Now that you have your data ready it is time to write the fusion pipeline. For that, you need to initialize your dataset and instantiate a **VDBVolume** object from our vdbfusion library. At this point, you should also pick a voxel size (resolution) and a truncation distance. The values chosen on this snippet are a good starting point.

NOTE: If you didn't follow the recommended **Dataset** API, your implementation might defer to the one below. You just need to pass numpy arrays to the **integrate** function.

```
1 from vdbfusion import VDBVolume
2
3 vdb_volume = VDBVolume(voxel_size=0.1, sdf_trunc=0.3, space_carving=False)
4
5 # You need to define your own Dataset, see snippet below
6 dataset = Dataset(...)
7
8 for scan, origin in dataset:
9     vdb_volume.integrate(scan, origin)
```

4. Visualizing the results

Done! You just built your map. Now it's time to inspect your results, for doing so I will be using Open3D but you can choose whatever library you like the most.



```
1 import open3d as o3d
2 # Extract a mesh from vdbfusion
3 vertices, triangles = vdb_volume.extract_triangle_mesh()
4
5 # Visualize the results
6 mesh = o3d.geometry.TriangleMesh(
7     o3d.utility.Vector3dVector(vertices),
8     o3d.utility.Vector3iVector(triangles),
9 )
10 mesh.compute_vertex_normals()
11 o3d.visualization.draw_geometries([mesh])
```

Please remember to report back the time spent on each of the 4 steps!