Homework 1: TSDF Fusion

姓名: 方嘉聪 学号: 2200017849

1 文件结构与最终效果

- 在`./results/`文件夹下存储有导出的结果文件:
  - `mesh_gray.ply` 和 `mesh_color.ply` 分别为灰度和彩色的 fusion 结果.
  - `mesh_first_frame.ply` 为第一帧的 fusion 结果, 用于确认 fusion 是否正确.
  - `pointcloud_*.ply` 为每 100 帧存储的点云数据.
- `demo.py` 与 `fusion.py` 为主要代码文件.

使用 MeshLab 可视化结果见 **Figure** 1 和 **Figure** 2.

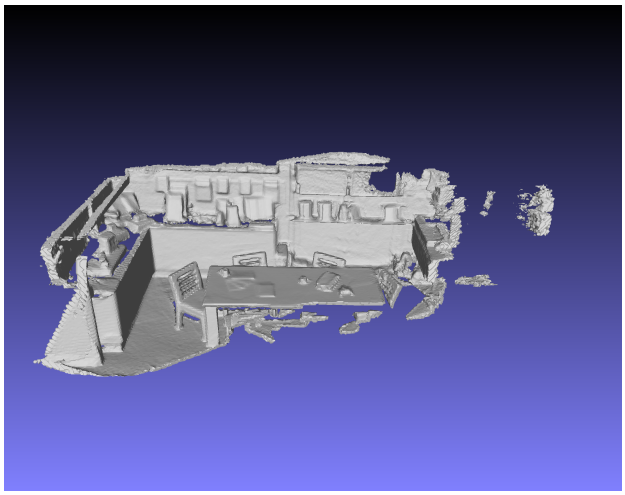

Figure 1: 灰度 TSDF Fusion 结果

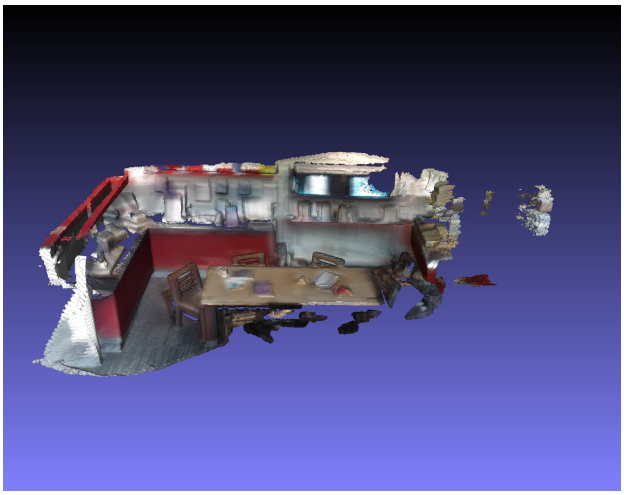

Figure 2: 彩色 TSDF Fusion 结果

2 Task 1: 从深度图生成点云

2.1 坐标转换

利用 `numpy.meshgrid` 生成网格, 而后左乘相机内参矩阵的逆矩阵, 逆变换到相机坐标系下, 再左乘相机外参矩阵, 变换到世界坐标系下. 最后将点云数据存储为 `.ply` 文件. 具体代码简要如下:

```
def cam_to_world(depth_im, cam_intr, cam_pose, im_index):
    H, W = depth_im.shape
    # Create a grid of pixel coordinates
    i, j = np.meshgrid(np.arange(H), np.arange(W), indexing="ij")
    pixels = np.stack([j, i, np.ones_like(i)], axis=-1).reshape(-1, 3)
    # Convert pixel coordinates to camera coordinates
    cam_intr_inv = np.linalg.inv(cam_intr)
    cam_cor = np.dot(cam_intr_inv, pixels.T).T
    cam_cor *= depth_im.flatten()[:, None]
    # Convert the camera coordinates to world coordinates
```

```
    world_pts = np.dot(
        cam_pose, np.hstack([cam_cor, np.ones((cam_cor.shape[0], 1))]).T
    )[:3, :].T
    if im_index % 100 == 0: # Save pointcloud every 100 frames
        pointcloud = trimesh.PointCloud(world_pts)
        pointcloud.export(f"pointcloud_{im_index}.ply")

    return world_pts
```

2.2 点云可视化

这里每隔 100 帧存储一次点云数据, 同时在 MeshLab 中打开, 可视化结果见 **Figure** 3.

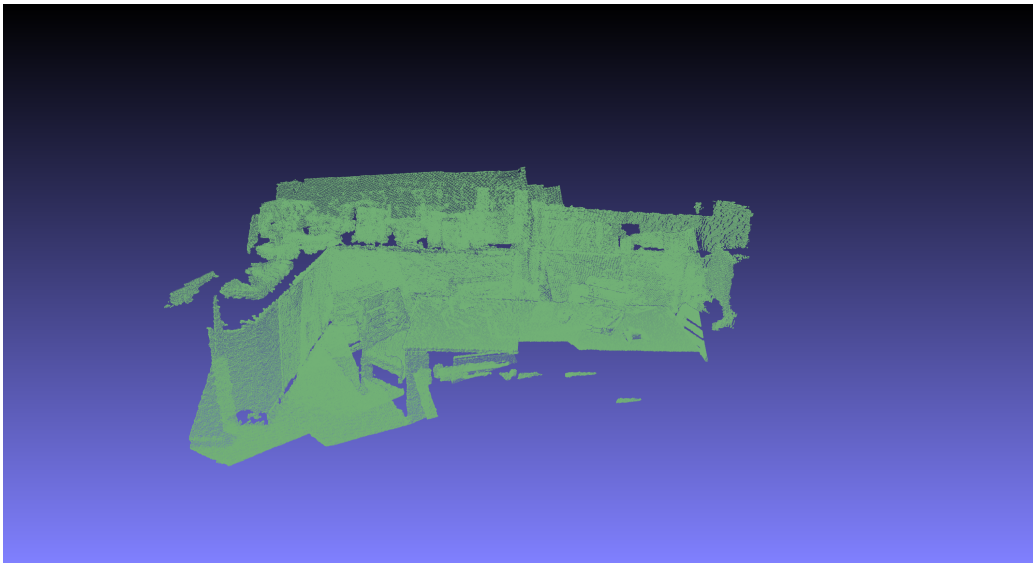

Figure 3: 每 100 帧存储的点云数据可视化 (MeshLab)

2.3 体素场范围

在 `demo.py` 中每一次迭代都更新体素场的范围即可, 使用 `numpy` 进行批量化操作.

```
max_xyz, min_xyz = np.max(view_frust_pts, axis=0), np.min(view_frust_pts, axis=0)
vol_bnds[:, 0] = np.minimum(vol_bnds[:, 0], min_xyz)
vol_bnds[:, 1] = np.maximum(vol_bnds[:, 1], max_xyz)
```

3 Task 2: 从深度图采样

3.1 体素场格点划分

在 `TSDFVolume` 类中, 通过 `np.round((vol_bnds[:, 1]-vol_bnds[:, 0])/voxel_size)` 得到体素的维度. 而后使用 `np.meshgrid` 在体素场中生成网格, 用于后续的体素场更新.

```
xv, yv, zv = np.meshgrid(
        np.arange(self.vol_dim[0]),
        np.arange(self.vol_dim[1]),
        np.arange(self.vol_dim[2]),
        indexing="ij",
    )
self.vox_coords = np.stack([xv, yv, zv], axis=-1).reshape(-1, 3)
```

3.2 采样点坐标转换

差不多是 **Task 1** 的逆过程, 依次经过: (需要进行矩阵的变换, 同样使用 `numpy` 进行批量化操作):

- 体素格点 → 世界坐标: 体素格点乘以体素大小, 加上体素场的左下角坐标.
- 世界坐标 → 相机坐标: 世界坐标左乘相机外参矩阵的逆矩阵.
- 相机坐标 → 图像坐标: 相机坐标左乘相机内参矩阵.

```
# voxel grid coordinates => world coordinates
world_coords = self.vox_coords * self.voxel_size + self.vol_bnds[:, 0]  # (N, 3)
world_coords_hom = np.hstack([world_coords, np.ones((world_coords.shape[0], 1))])

# world coordinates => camera coordinates (N, 3)
cam_coords = (np.dot(np.linalg.inv(cam_pose), world_coords_hom.T).T)[:, :3]

# camera coordinates => pixel coordinates
pixel_coords = np.dot(cam_intr, cam_coords.T).T  # (N, 3)
pixel_coords[:, 0] /= pixel_coords[:, 2]
pixel_coords[:, 1] /= pixel_coords[:, 2]
```

3.3 深度采样

将 `pixel_coords` 的坐标取整, 用于索引深度图, 从而得到深度值. 此外使用 mask 将超出深度图范围的点剔除.

```
# Round to nearest pixel
pixel_coords = np.round(pixel_coords[:, :2]).astype(int)
valid_pix = (   # Mask out-of-bound coordinates
    (pixel_coords[:, 0] >= 0) & (pixel_coords[:, 0] < W) \
    & (pixel_coords[:, 1] >= 0) & (pixel_coords[:, 1] < H)
)
# Sample depth values
depth_val = np.zeros(self.vox_coords.shape[0])
depth_val[valid_pix] = depth_im[
    pixel_coords[valid_pix, 1].astype(int),
    pixel_coords[valid_pix, 0].astype(int),
]
```

注: 应该也可以使用双线性插值, 这里为了方便直接取整. 观察最终的效果可以接受.

4 Task 3: 计算单帧 TSDF

根据每个采样点的深度值, 计算其与体素格点的距离, 并更新体素场的 TSDF 值, 同时进行截断操作.

$$\text{TSDF} = \min\{1.0, (\texttt{depth\_val} - z)/\texttt{self.trunc\_margin}\}$$

在这里定义了一个 mask 来排除一些异常值, 具体如下:

```
depth_sample = np.zeros(self.vox_coords.shape[0])
depth_sample[valid_pix] = (
    depth_val[valid_pix]
    - cam_coords[valid_pix, 2]  # depth of the voxel grid coordinates
)
# Mask to exclude points outside the truncation margin and invalid depth values
valid_mask = (depth_val > 0) & (depth_sample >= -self.trunc_margin)
old_color_weight = self.weight_vol[valid_mask]
tsdf = np.minimum(1.0, depth_sample / self.trunc_margin)
```

5 Task 4: 融合多帧 TSDF

5.1 融合 TSDF

加权平均更新体素场的 TSDF 值和权重值即可, 依旧使用在 **Task 3** 中定义的 mask 来排除一些异常值. 具体如下:

```
self.tsdf_vol[valid_mask] = (
        self.tsdf_vol[valid_mask] * self.weight_vol[valid_mask]
        + tsdf[valid_mask] * obs_weight
) / (self.weight_vol[valid_mask] + obs_weight)
self.weight_vol[valid_mask] += obs_weight
```

为了检验是否融合正确, 除了保存最终的 Fusion 结果 (见 **Figure** 1), 还保存了第一帧的 Fusion 结果, 见 **Figure** 4.

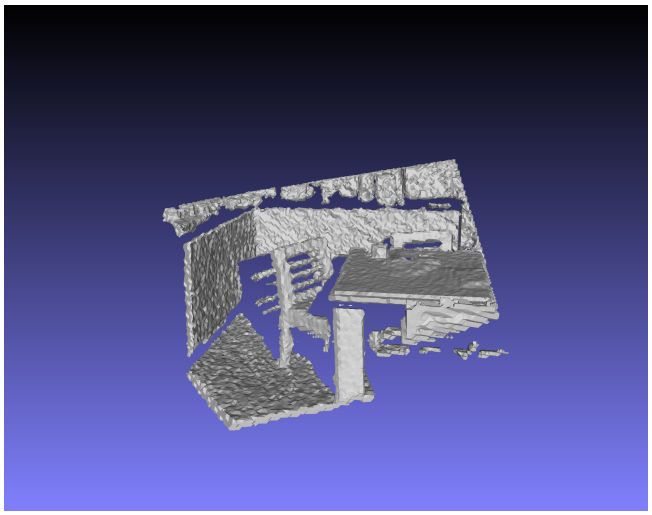

Figure 4: 第一帧的 TSDF Fusion 结果

运行的 Average FPS 为 2.11.

5.2 调用 Marching Cubes 生成 Mesh

在 `TSDFVolume` 类中定义 `get_mesh` 方法. 先 reshape 体素场的 TSDF 值, 再调用 `skimage.measure` 的 `marching_cubes` 生成点, 面和法向量. 注意这里得到的点为体素格点, 需要乘以体素大小, 加上体素场的左下角坐标, 转换到世界坐标系下. 而后使用 `trimesh` 生成 Mesh, 并保存为 `.ply` 文件.

```
def save_mesh(self, filename, vertex_colors=None):
    tsdf_vol_vis = np.copy(self.tsdf_vol).reshape(self.vol_dim)
    verts, faces, norms, vals = measure.marching_cubes(tsdf_vol_vis, level=0)
    verts = verts * self.voxel_size + self.vol_bnds[:, 0]

    mesh = trimesh.Trimesh(verts, faces, vertex_normals=norms)
    mesh.export(f"{filename}.ply")
```

6 Extra Task: 带颜色的 TSDF Fusion

我选择使用图片的颜色信息，实现带颜色的 TSDF Fusion. 类似对于深度的采样, 颜色有三个通道. 具体流程如下:

- **初始化:** 在 `TSDFVolume` 类中增加 `color_vol` 用以存储颜色信息, 初始化为 0.

  ```
  self.color_vol = np.zeros(
      (self.vol_dim[0] * self.vol_dim[1] * self.vol_dim[2], 3),
      dtype=np.float32,
  )
  ```

- **颜色采样**: 在 `integrate` 方法中增加对于颜色信息的融合, 类似深度的采样, 使用 `pixel_coords` 的坐标取整, 用于索引颜色图, 从而得到颜色值.

  ```
  color_val = np.zeros((self.vox_coords.shape[0], 3))
  color_val[valid_pix] = color_im[
      pixel_coords[valid_pix, 1].astype(int),
      pixel_coords[valid_pix, 0].astype(int),
  ]
  ```

- **多帧颜色融合:** 使用加权平均更新体素场的颜色值, 实际实现中使用与和深度融合相同的 mask 及累计的权重值 `old_color_weight = self.weight_vol[valid_mask]`

  ```
  self.color_vol[valid_mask] = (
      self.color_vol[valid_mask] * old_color_weight[:, None]
      + color_val[valid_mask] * obs_weight
  ) / self.weight_vol[valid_mask][:, None]
  self.color_vol = np.clip(self.color_vol, 0, 255)
  ```

- **Mesh 生成:** 在 `get_mesh` 方法中, 生成 Mesh 时, 传入颜色信息. 注意需要使用点索引得到对应的颜色信息.

  ```
  verts_index = np.round(verts).astype(int)  # use to index color_vol
  vertex_colors = vertex_colors.reshape(
      self.vol_dim[0], self.vol_dim[1], self.vol_dim[2], 3
  ).astype(np.uint8)
  ```

```
vertex_colors = vertex_colors[
    verts_index[:, 0], verts_index[:, 1], verts_index[:, 2]
]
mesh = trimesh.Trimesh(
    verts, faces, vertex_normals=norms, vertex_colors=vertex_colors
)
```

最终的效果见 **Figure** 2.