3D-Vision

Camera Model

Pinhole Camera Model

- Assume f is the focal length, $\mathbb{O}$ is the aperture, also the pinhole, also the center of the camera. (x, y, z) is the object position in the camera coordinate system, where z is measured along the optical axis. (x', y') is the image position in the image plane.
- The pinhole camera model is given by:

$$x' = f\frac{x}{z} \tag{1}$$
$$y' = f\frac{y}{z} \tag{2}$$

- limitation: pinhole camera requires the aperture to be infinite small to get a unblurry image, but now the light will be too weak.

Lens Camera Model

- Assume f is the focal length, (x, y, z) is the object position in the camera coordinate system, where z is measured along the optical axis. (x', y', z') is the image position in the image plane.
- the lens camera model is given by:

$$x' = z'\frac{x}{z} \tag{3}$$
$$y' = z'\frac{y}{z} \tag{4}$$
$$z' = z \tag{5}$$

- limitations:
  - depth of field: only objects with specific z can have a clear image.
  - Radial Distortion: Deviations are most noticeable for rays that pass through the edge of the lens. (noticeable for camera with wide angle lens, like fish-eye lens)
    - Cushion distortion
    - Barrel distortion

Perspective Camera Model

Intrinsics

- important formulas:

$$(x, y, z) \to (f\frac{x}{z} + c_x, f\frac{y}{z} + c_y)$$

  if we care about pixel coordinates,

$$(x', y', z') \to (k \cdot (f\frac{x}{z}) + c_x, l \cdot (f\frac{y}{z}) + c_y) = (\alpha\frac{x}{z}, \beta\frac{y}{z})$$

  introduce homogeneous coordinates,

$$z\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \alpha & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

  , simply writen as

$$P' = MP = K[I, 0]P$$

  where K is the intrinsic matrix, I is the identity matrix, and P is the world-coordinate position of the object.
- if we consider the case where the image coordinates are not perfectly perpendicular, specifically, the angle between u-axis and v-axis is θ, θ(is 90 for most industry-level cameras), then we have instrinsic matrix

$$K = \begin{bmatrix} \alpha & -\alpha cot\theta & c_x \\ 0 & \frac{\beta}{sin\theta} & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Extrinsics

- important formula:

$$P_camera = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} P_world$$

  where R is the rotation matrix, t is the translation vector.
- include both intrinsic and extrinsic parameters in the camera matrix,

$$P = K \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} P_world$$

- write the projective transformation in a vector form:

$$\begin{aligned} P'_{3\times1} &= MP_w = K_{3\times3}[R,T]_{3x4}P_{w4\times1} \\ &= \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} P_w = \begin{bmatrix} m_1P_w \\ m_2P_w \\ m_3P_w \end{bmatrix} \end{aligned}$$

  so the projected pixel is

$$(\frac{m_1P_w}{m_3P_w}, \frac{m_2P_w}{m_3P_w})$$

  .

Orthographic Camera Model

- Assume f is the focal length, (x, y, z) is the object position in the camera coordinate system, where z is measured along the optical axis. (x, y) is the image position in the image plane.

Camera Calibration

- Goal: Estimate the intrinsic and extrinsic parameters of the camera from one or multiple images.
- DOF:
  - rotation: 3DOF
  - translation: 3DOF
  - intrinsic parameters: 2DOF(α, β) + 2DOF(c_x, c_y) + 1DOF(θ) = 5DOF
- we need at least 6 corresponding pairs, each pair provide two functions $x \to u$ and $y \to v$. In practice, using more than 6 correspondences enables more robust results.

```
NOTE: the correspondences cannot be taken from the same plane, because during the calculation process, we don't have
```

method

Given camera model

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{m_1P_i}{m_3P_i} \\ \frac{m_2P_i}{m_3P} \\ 1 \end{bmatrix}$$

then we have

$$u_i(m_3P_i) - m_1P_i = 0 \tag{6}$$
$$v_i(m_3P_i) - m_2P_i = 0 \tag{7}$$

if we take n points, then we have a homogeneous Linear System:

$$\begin{bmatrix} P_1^T & 0^T & -u_1 P_1^T \\ 0^T & P_1^T & -v_1 P_1^T \\ \vdots & \vdots & \vdots \\ P_n^T & 0^T & -u_n P_n^T \\ 0^T & P_n^T & -v_n P_n^T \end{bmatrix}_{2n\times 12} \begin{bmatrix} m_1^T \\ m_2^T \\ m_3^T \end{bmatrix}_{12\times 1} = 0$$

we add constrain

$$|m|^2 = 1$$

to avoid trivial solution. Then we can simply do SVD on the left matrix and the last column of V will be the solution $\hat{M} = (m_1, m_2, m_3)$. Now we need a scaling factor ρ to restore the original M:

$$M = \rho\hat{M} = \begin{bmatrix} \alpha r_1^T - \alpha cot\theta r_2^T + c_x r_3^T & \alpha t + x - \alpha cot\theta t_y + c_x t_z \\ \frac{\beta}{sin\theta} r_2^T + c_y r_3^T & \frac{\beta}{sin\theta} t_y + c_y t_z \\ r_3^T & t_z \end{bmatrix} = K[R, T]$$

where $R = (r_1, r_2, r_3)_{3\times 3}, T = (c_x, c_y, c_z)^T$. The solution is:

Calibration Problem

$$M = \rho\hat{M} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha\cot\theta \boldsymbol{r}_2^T + c_x \boldsymbol{r}_3^T & \alpha t_x - \alpha\cot\theta t_y + c_x t_z \\ \frac{\beta}{\sin\theta}\boldsymbol{r}_2^T + c_y \boldsymbol{r}_3^T & \frac{\beta}{\sin\theta} t_y + c_y t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix} = \mathrm{K}[\mathrm{R} \quad \mathrm{T}]$$

$$\mathrm{K} = \begin{bmatrix} \alpha & -\alpha\cot\theta & c_x \\ 0 & \frac{\beta}{\sin\theta} & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

A $\quad$ $\mathbf{b}$

Box 1

$$\hat{\mathrm{A}} = \begin{bmatrix} \hat{\mathbf{a}}_1^{\mathrm{T}} \\ \hat{\mathbf{a}}_2^{\mathrm{T}} \\ \hat{\mathbf{a}}_3^{\mathrm{T}} \end{bmatrix} \quad \hat{\mathrm{b}} = \begin{bmatrix} \hat{\mathrm{b}}_1 \\ \hat{\mathrm{b}}_2 \\ \hat{\mathrm{b}}_3 \end{bmatrix}$$

Estimated values from $\hat{M}$

Intrinsic

$$\rho = \frac{\pm 1}{|\hat{\mathbf{a}}_3|} \qquad \begin{aligned} c_x &= \rho^2(\hat{\mathbf{a}}_1 \cdot \hat{\mathbf{a}}_3) \\ c_y &= \rho^2(\hat{\mathbf{a}}_2 \cdot \hat{\mathbf{a}}_3) \end{aligned}$$

$$\cos\theta = \frac{(\hat{\mathbf{a}}_2 \times \hat{\mathbf{a}}_3) \cdot (\hat{\mathbf{a}}_3 \times \hat{\mathbf{a}}_1)}{|\hat{\mathbf{a}}_2 \times \hat{\mathbf{a}}_3| \cdot |\hat{\mathbf{a}}_3 \times \hat{\mathbf{a}}_1|}$$

74

Calibration Problem

$$M = \rho\hat{M} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha\cot\theta \boldsymbol{r}_2^T + c_x \boldsymbol{r}_3^T & \alpha t_x - \alpha\cot\theta t_y + c_x t_z \\ \frac{\beta}{\sin\theta}\boldsymbol{r}_2^T + c_y \boldsymbol{r}_3^T & \frac{\beta}{\sin\theta} t_y + c_y t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix} = \mathrm{K}[\mathrm{R} \quad \mathrm{T}]$$

$$\mathrm{K} = \begin{bmatrix} \alpha & -\alpha\cot\theta & c_x \\ 0 & \frac{\beta}{\sin\theta} & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

A $\quad$ $\mathbf{b}$

Box 1

$$\hat{\mathrm{A}} = \begin{bmatrix} \hat{\mathbf{a}}_1^{\mathrm{T}} \\ \hat{\mathbf{a}}_2^{\mathrm{T}} \\ \hat{\mathbf{a}}_3^{\mathrm{T}} \end{bmatrix} \quad \hat{\mathrm{b}} = \begin{bmatrix} \hat{\mathrm{b}}_1 \\ \hat{\mathrm{b}}_2 \\ \hat{\mathrm{b}}_3 \end{bmatrix}$$

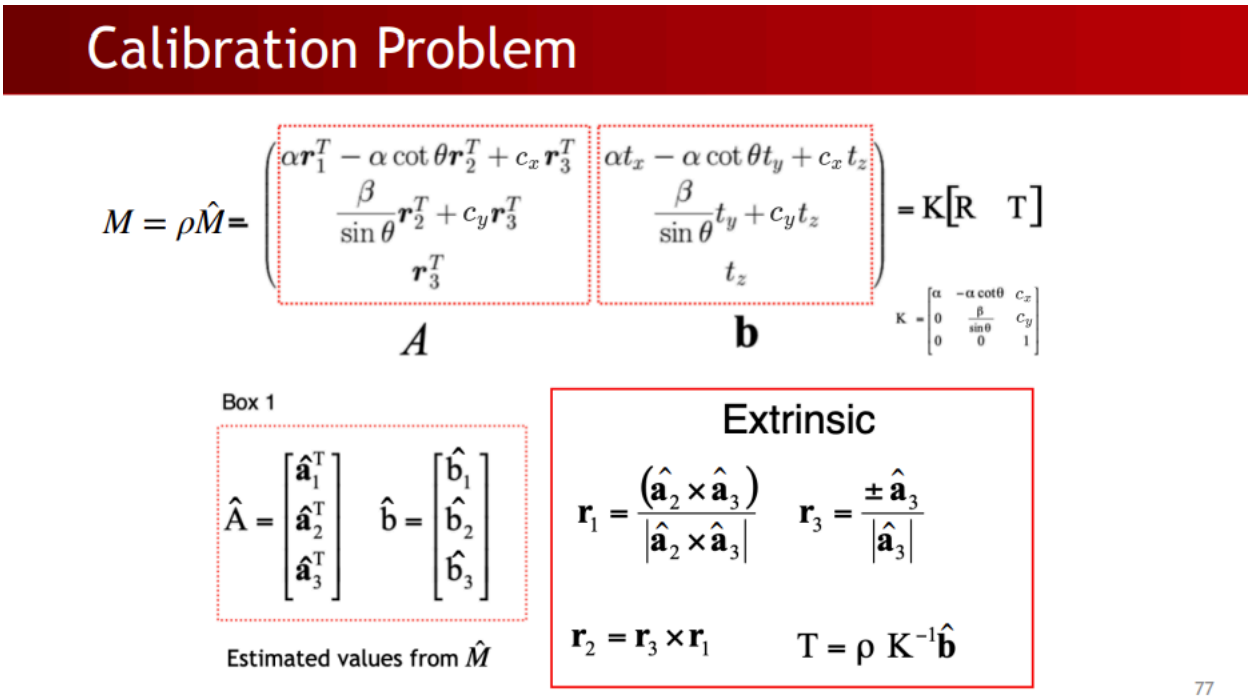
Estimated values from $\hat{M}$

Intrinsic

$$\alpha = \rho^2 |\hat{\mathbf{a}}_1 \times \hat{\mathbf{a}}_3| \sin\theta$$

$$\beta = \rho^2 |\hat{\mathbf{a}}_2 \times \hat{\mathbf{a}}_3| \sin\theta$$

76



depth images

Depth

- z-depth: distance along z-axis(optical axis) from the optical center to the point
- ray-depth: the distance between the optical cener and the point
- depth image: a 2D image where each pixel represents the z-depth of the corresponding point in the 3D space. depth image is a 2.5D representation, because it cannot enable 3D distance measurement by itself, it need the camera intrinsic to convert 2D image coordinates to 3D space.

Depth Backprojection

For a depth image with intrinsic K, and a pixel (u, v, z), we have

$$u = \alpha\frac{x}{z} + c_x \tag{8}$$
$$v = \beta\frac{y}{z} + c_y \tag{9}$$

then we can solve

$$x = z\frac{u - c_x}{f_x} \tag{10}$$
$$y = z\frac{v - c_y}{f_y} \tag{11}$$

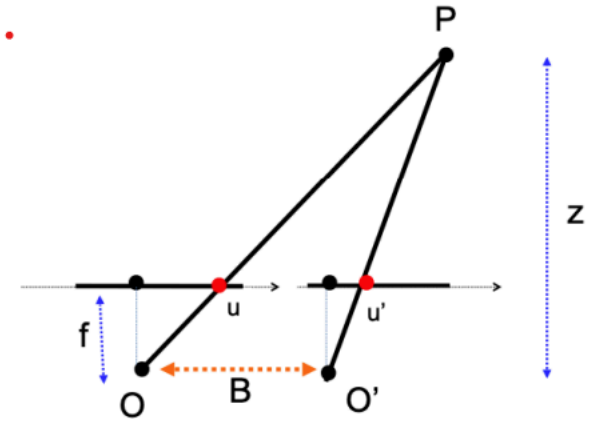
to get the corresponding 3D point in the camera coordinate system.

Depth Sensor

- Measure multi-point distance information across a wide Field-of View(FoV)

Stereo Sensors

1. first calculate disparity.



$$u - u' = \frac{B \cdot f}{z} = \text{disparity} \qquad \text{[Eq. 1]}$$

Note: Disparity is inversely proportional to depth

the image is actually generated behind the lens, here we move the image to the front in order to find geometric relationships. And we can notice that the depth we get is actually the depth to the lens not to the image plane.

2. Second we find disparity maps. We have two depth cameras, which gives us two images from different viewpoints. In this step, we find the correspondence in this step in pixel level and use step1 to calculate the depth. **NOTE:** this depth is mapped to a pixel in the depth image(we can choose any one of the two depth image as a reference), but it's not the depth of pixel at the same position in RGB image.
3. dpeth-to-RGB registration: use depth to get the pointcloud, transfer the pointcloud to the RGB camera using relative extrinsics, then project the pointloud to image plane and sample the get depth of each pixel in the RGB image.
4. pros and cons:
    - Advantages:
        a. Robust to the illumination of direct sunlight
        b. Low implementation cost
    - Disadvantage:
        a. Finding correspondences along and is hard and erroneous
            - textureless surface
            - occlusion, repitition
            - specularities, transparency

Active Stereo

Directly project predefined image to the scene(IR for Intel Realsense). It is used to solve the problem of finding correspondence, but it's still hard to deal with transparency or specularities. Black surface can also be a problem because it absorbs the light and no correspondence can be found.

More 3D representation

regular form:

- multi-view images
- depth
- volumetric

irregular form:

- surface mesh
- point cloud
- implicit representations

---

mesh

how to store a mesh

1. STL(Triangle List):
   - store information: Face(3 positions)
   - no connectivity information
2. OBJ(Object File):
   - store information: Vertex(3 positions), Face(3 vertex indices)
   - Convention is to save vertices in counterclockwise order (right hand rule) for normal direction (pointing out)

Compute Mesh Geodesic Distance

1. method 1: find the shortest path(along edges) between two points in the mesh.
2. method 2: Fast Marching
3. method 3: MMP

point cloud

- Irregular and orderless data
- A light-weight geometric representation
- Compact to store
- Easy to understand and generally easy to build algorithms

how to sample point cloud from a surface

we usually sample point cloud from a mesh surface, but we don't reconstruct the surface using the point cloud.

uniform sampling

1. Compute the areas of each individual face
2. Compute the probability of each face and use it as weight
3. Independent identically distributed (i.i.d.) sample faces according to the weights
4. For each sampled face, uniformly sample from one triangle face
   - General case: for a triangle with vertices : $v1, v2, v3$ we have $x = v3 + a1(v1 - v3) + a2(v2 - v3) = a1v1 + a2v2 + (1 - a1 - a2)v3$, where $a1$ and $a2$ are uniform variates in the interval $[0, 1]$. If $a1 + a2 \leq 1$, then x will be inside the triangle (or on the edges); if $a1 + a2 > 1$, then x can be mapped back to the triangle interior via $x = (1 - a1)v1 + (1 - a2)v2 + (a1 + a2)v3$

farthest point sampling

the exact solution will be a NP-hard problem, but we can approximate it using a greedy algorithm.

1. Over sample the shape by any fast method (e.g. uniformly sample N=10000 samples)
2. iteratively select K points:
   - at the first iteration, randomly select one point
   - for each subsequent iteration, find the point farthest from the previous k points

distance metrics for point cloud

1. chamfer distance:

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} min_{y \in S_2} ||x - y||_2 + \sum_{y \in S_2} min_{x \in S_1} ||y - x||_2$$

2. Earth Mover's distance:

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \to S_2} \sum_{x \in S_1} ||x - \phi(x)||_2$$

   where ϕ is a bijection
3. chamfer distance is insensitive to sampling, but EMD is sensitive to sampling.

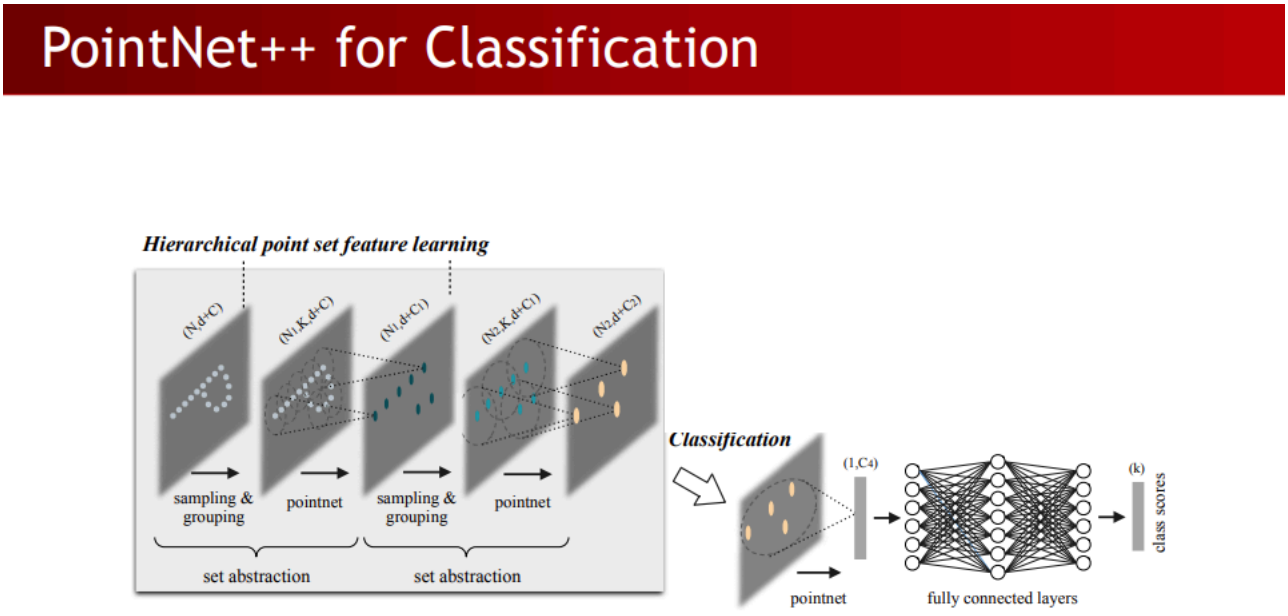
3D Deep Learning

PointNet

- essence: we need a network, that can gives out local point features and global features. Specifically, we don't want the permutation of the points to influence the output.
- architecture:
  use sum, mean or max pooling to aggregate the local features into global features.
- merits:
  robustness to many kind of data corruption:
  - missing data: a net trained on pointcloud of 1024 points still works on pointcloud of 512 points. Why? Because the aggregation function(max), just takes out the important points, as long as theses points are present, the outcome won't change much.
  - resolution: the network weights doesn't care about the number of points in a pointcloud. This makes it promising to be deployed in some resource-limited environments.
  - occlusion
  - noise
- limitations:
  - No local context for each point
  - Global feature depends on absolute coordinate. Hard to generalize to unseen scene configurations!

PointNet++

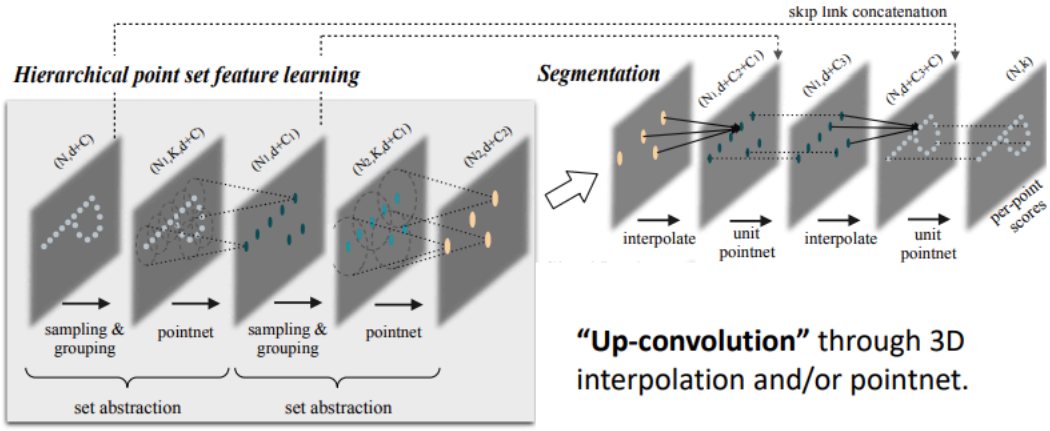
- essence: Recursively apply PointNet to local regions of the pointcloud, and then aggregate the features.
  - Hierarchical feature learning: learn local features at multiple scales, and then aggregate them into global features.
  - Local translation invariance: local points are centered before calculate features in order to leave out the reliance on global coordinates.
  - Permutaion invariance: thanks to the logic of PointNet
- classification:



C is the PointNet feature dim, d = 3 (append the original x,y,z to local geometry feature)

- segmentation:

PointNet++ for Segmentation



3D interpolation from N_2 to N_1 $(N_1 > N_2)$: 1) for the existing points in N_2, keep its C_2-dim feature;
2) for the other points, do 3-interpolate (inverse distance weighted average based on 3 nearest neighbors);
3) for all points, concatenate the original (x,y,z) and skip link (concatenate) from the feature (C_1-dim) from the left side N_1 layer.

3D ConvNet

sparse convolution

PointNet++ takes local feature into consideration, but all the points are processed in the ball are processed with the same network, which means PointNet++ is a isotropic to the points around the seed point. However, in a 2D conv network, the conv kernel has different value for different position, which makes conv network more representative of the local context.

3D conv kernel takes voxels for convolution, which makes the resolution of voxels, and where to place voxels really important.

- sparse conv: only place voxels at the surface of the object(we can reconstruct the voxels using the point cloud), and then apply the conv kernel only to these voxels.
- resolution: voxelization definitely causes information loss, because it discretize the whole space. Specifically, one voxel has hold many points in a point cloud.
  - we alleviate this problem by using the average coordinate of the points in the voxel as a feature.
  - for large scale scene, like self-driving car, the info loss caused by resolution is not really an issue.
  - consider the real sensors has certain errors, the info loss caused by voxelization is acceptable.

pros and cons of sparse conv:

- Pros:
  - A way higher efficiency than dense conv
  - Regular grid that supports indexing(faster to find neighbors than FPS in PointNet++)
  - Similarly expressive compared to 2D Conv
  - Translation equivariance similar to 2D Conv
- Cons:
  - Discretization error(information loss)

Sparse Conv vs. Point Cloud Networks

- Sparse Conv:
  - +: Kernels are spatial anisotropic
  - +: More efficient for indexing and neighbor query
  - +： suitable for large-scale scenes
  - -: limited resolutions
- Point cloud networks:
  - +: high resolution
  - +: easier to use and can be the first choice for a quick try
  - -: slightly lower performance
  - -: slower if performing FPS and ball query
- we prefer sparse conv for large-scale scenes, and point cloud networks for small-scale scenes.