BuildAnyPoint: 3D Building Structured Abstraction from Diverse Point Clouds

Supplementary Material

7. More Technical details

7.1. Training Details

Diffusion models are primarily implemented based on the sparse VAE and diffusion structure provided by XCube [36]. For VAE and latent diffusion training, we use 4 NVIDIA A40 GPUs. The training objectives for the two VAEs—dense and sparse geometry—are unified as:

$$\begin{aligned}\mathcal{L}_{\text{VAE}} \quad &= \mathbb{E}_{\{\mathcal{G},A_N\}}[\mathbb{E}_{X\sim p(\mathcal{E})}[\lambda_0 \text{BCE}(\mathcal{G},\tilde{\mathcal{G}})+ \\ &\lambda_1\mathcal{L}_1(A_N,\tilde{A}_N)] + \lambda_2\mathbb{KL}(p_{\mathcal{E}}(X) \parallel p(X))],\end{aligned} \tag{7}$$

where $\text{BCE}(\cdot)$ represents the binary cross-entropy for grid occupancy, $\mathcal{L}_1$ denotes the L1 loss, and $\mathbb{KL}(\cdot \parallel \cdot)$ is the KL divergence. In practice, we set the weighting factors to be $\lambda_0 = 20, \lambda_1 = 50, \lambda_2 = 0.03$. We train the VAE for $\mathcal{G}_d$ and $\mathcal{G}_s$ with 28120 and 71660 iterations, respectively. The training procedure for 3D latent diffusion follows the same structure implemented by [36]. For raw input conditioning, we use an additional point encoder to quantize the incomplete point cloud to a voxel grid and concatenate it with the latent feature. We set the diffusion step to be 100 with a linear noise schedule, and trained for 6500 iterations.

Autoregressive transformer is implemented based on MeshAnything V2 [9] to minimize the negative log-likelihood of the ground-truth mesh token sequence conditioned on the input point cloud. The training loss is defined as the cross-entropy loss over the entire mesh token sequence:

$$\mathcal{L} = -\sum_{t=1}^{L} \log P\left(t_t = \mathcal{T}_M^{(t)} \mid \mathcal{T}_P, \mathcal{T}_M^{<t}; \phi\right), \tag{8}$$

where L is the length of the target mesh token sequence, $\mathcal{T}_M^{(t)}$ is the ground-truth token at position t, $\mathcal{T}_M^{<t}$ denotes all tokens before t, $\mathcal{T}_P$ is the tokenized point cloud condition, and ϕ represents the parameters of the decoder-only transformer. We implement our model using the OPT-350M architecture as the backbone transformer. The model is trained for 520000 iterations on 4 NVIDIA A40 GPUs with a total batch size of 8. We freeze the point cloud encoder, which has been trained on large-scale daily object datasets and thus is capable of handling complex geometries.

7.2. Training Data Processing

Normalization. To ensure consistent scale and orientation across all training samples, we implement a robust normalization procedure. The point cloud is first centered by translating its centroid to the origin of the coordinate system. The translation vector is computed as the arithmetic mean of all point coordinates. Following translation, we apply uniform scaling based on the maximum extent along any coordinate axis, normalizing the point cloud to fit within the [-1, 1] range in all dimensions. This normalization strategy preserves aspect ratios while eliminating scale variations that could impede model learning.

Normal Estimation. We compute surface normals using a k-nearest neighbors approach with 30 neighboring points, providing crucial information about local surface orientation. These normals are essential for capturing fine geometric details and enabling more meaningful latent space representations in the VAE.

Multi-Resolution Voxelization. The core of our preprocessing pipeline involves converting the normalized point clouds into structured volumetric representations. We employ a sparse voxelization approach that efficiently represents 3D space while minimizing memory requirements. The process begins with an initial voxelization at low resolution of 128^3, which serves as the foundation for generating multiple resolution levels, then moves to higher resolutions of up to 512^3. For each target resolution (*i.e.*, 128^3 and 512^3), the voxel centers are transformed to world coordinates and normalized to match the convolutional occupancy networks scale convention, ensuring compatibility with standard 3D deep learning architectures. This multi-resolution approach enables training of hierarchical models that can capture both global structure and local details.

Feature Splatting and Aggregation. To transfer point-level features to the voxel grid, we implement trilinear splatting operations. This technique distributes point features (normals) to neighboring voxels using trilinear interpolation weights, creating smooth feature fields that preserve spatial relationships. The splatted features are subsequently normalized to unit length, ensuring numerical stability during network training. This process results in two key data components: voxelized geometry and surface normal fields.

8. Metric Calculation Details

Our evaluation encompasses both the final mesh and the intermediate point cloud outputs. It is noteworthy that the accuracy metrics for the mesh (such as Chamfer Distance) are derived from point clouds sampled from its surface. Furthermore, the assessment of both point clouds and meshes involves a series of heuristic coordinate alignment transformations to ensure the fairness of the computations.

8.1. Data Processing

Mesh Preprocessing and Surface Sampling. The process begins with point cloud sampling from the mesh surface. To generate a uniformly distributed point cloud from the three-dimensional mesh, we employ a sampling strategy based on face area weighting. Initially, each face is examined; those with fewer than three vertices or invalid indices are disregarded. For polygonal faces, a fan triangulation method is applied to subdivide them into multiple triangles, ensuring subsequent sampling is performed on standard triangular elements. For each valid triangle, its area is computed and used as the weighting factor for sampling. The cumulative area of all triangles constitutes the total surface area, which in turn determines the probability of each triangle being selected. This area-weighted approach ensures that larger triangles have a higher likelihood of being sampled, thereby promoting an overall uniform distribution of points across the mesh surface and preventing local density variations due to over- or under-representation of small faces.

After establishing the triangle selection probabilities, each sampling iteration randomly selects a triangle, and a point is generated within its interior using uniformly distributed barycentric coordinates. For each sampled point, two random numbers are generated and subjected to a necessary folding operation to guarantee that the point lies within the triangle boundaries rather than outside. This process is repeated until the desired number of points is attained, resulting in a uniformly distributed point cloud on the mesh surface.

In cases where the mesh contains no valid faces or has an insufficient total surface area, the method defaults to randomly selecting points directly from the vertex set. This fallback mechanism ensures the robustness of the sampling algorithm and prevents failures when processing degenerate or empty meshes. Through this area-weighted sampling strategy, high-quality, uniformly distributed, and highly representative point cloud data can be generated from complex three-dimensional meshes.

Alignment Procedure. To obtain a robust initial pose, we first center the source and target point clouds by translating them to their respective centroid coordinate systems. Subsequently, we compute the covariance matrices for both centered point clouds and perform eigen-decomposition to extract the Principal Component Analysis (PCA) principal directions as an intrinsic coordinate system representing the shape. After arranging the eigenvectors in descending order of their corresponding eigenvalues and ensuring right-handed coordinate system consistency, we construct a rotation matrix that aligns the PCA coordinate system of the source point cloud to that of the target point cloud, using the target's eigenvectors as reference. The rotation matrix is further regularized via Singular Value Decomposition (SVD) to satisfy orthogonality constraints and maintain a positive unit determinant. The translation vector is then derived by matching the centroid positions of the two point clouds, thereby forming the initial rigid transformation matrix. This PCA-based initialization effectively reduces the search space caused by rotational and scale inconsistencies, enabling subsequent iterations to converge from a more reasonable starting state.

Following initial alignment, we refine the transformation using the Iterative Closest Point (ICP) framework implemented in Open3D, which is scale-sensitive. To enhance robustness, we first estimate normal vectors for both point clouds and employ adaptive thresholds that dynamically adjust the nearest-neighbor search range based on the scale of the point clouds. During iterative optimization, we prioritize the point-to-plane error metric to mitigate noise interference and revert to point-to-point ICP when convergence is insufficient, thereby ensuring algorithmic stability. Finally, by applying the calculated transformation to the predicted point cloud, we obtain a result aligned with the target point cloud.

8.2. Formulations

- **Number of vertices, faces, and planes (#V, #F and #P)** The number of vertices and faces is determined through direct counting upon loading the meshes. However, extracting geometrically distinct planes requires an effective algorithm. To identify distinct planes within the mesh, we begin by extracting the plane parameters corresponding to each face. For a given face, three vertices are selected, and two edge vectors are constructed, from which the face normal vector is derived. If the magnitude of this normal vector is below a negligible threshold, the face is considered degenerate and skipped. For valid faces, the normal vector is normalized to obtain a consistent and stable unit normal. Combined with an arbitrary vertex from the face, the plane offset is computed, thereby forming a complete geometric description of the plane associated with the face. After generating the plane parameters for all faces, we proceed to determine whether these planes geometrically represent the same planar surface. Specifically, each newly generated plane is iteratively compared against all previously recorded planes. The comparison first examines whether the normal vectors of the two planes point in the same or opposite direction—if the normals are highly aligned, the planes are considered parallel. Only under the condition of parallelism is the offset difference between the two planes evaluated. If this difference falls within a predefined threshold, the two faces are regarded as lying on the same geometric plane, and the new plane is not counted as distinct. If no matching plane is found within the existing set, the current plane is added as a new unique plane.
- **Failure Rate (FR %)**

City3D: The determination of a successful mesh is based on a twofold criterion, as implemented in the final validation step of its official repository. The process is deemed successful only if the core reconstruction algorithm returns a positive status and the resulting 3D model contains at least one geometric facet. Specifically, a boolean status flag is first obtained from the reconstruct method, which signifies whether the internal optimization process, incorporating the novel roof preference energy term and hard constraints for topology, converged to a valid solution. Subsequently, the resulting Map data structure is checked for geometric content; a non-zero number of facets confirms that a tangible polygonal mesh was generated. If both conditions are satisfied, the model is serialized to disk. Conversely, if either the algorithm reports a failure (status is false) or produces an empty model, the entire reconstruction is considered to have failed, triggering a runtime exception.

Point2Building employs an iterative, validation-driven generation process to ensure the structural plausibility of reconstructed meshes (refers to Appendix B in [28]). Upon generating a candidate set of vertices, the model proceeds to generate the connecting faces. The resulting mesh hypothesis is then subjected to a series of hard-coded geometric checks. These include verifying the presence of a stop token, assessing the coverage of the input point cloud's footprint by the ground polygon, ensuring proper connectivity between walls and the ground edges, and checking for invalid diagonal edges on walls. A mesh is accepted only if it passes all checks.

BuildAnyPoint: The post-generation validation process starts by verifying completion through a stop token indicator and identifying any invalid faces containing NaN values. The validated mesh is then saved, labeled with a success or failure designation based on the initial completion check.

- **Chamfer Distance (CD)** between predicted (A) and ground truth (B) point clouds using the L1 (Manhattan) distance metric. Both point clouds are first uniformly sampled to a fixed size of 16384 points to ensure consistent comparison:

$$\begin{aligned} CD(A,B) \quad &= \frac{1}{|A|}\sum_{a\in A}\min_{b\in B}\|a-b\|_1 + \\ &\frac{1}{|B|}\sum_{b\in B}\min_{a\in A}\|b-a\|_1 \end{aligned} \tag{9}$$

- **F-score** is computed to evaluate the consistency between two point clouds. Both point clouds are uniformly sampled to a fixed number similar to the calculation of CD. For a predefined distance threshold d, precision $P(d)$ is defined as the proportion of points in A whose nearest neighbor in B lies within d, while recall $R(d)$ is the proportion of points in B whose nearest neighbor in lies within d. The F-score is the harmonic mean of precision and recall:

$$\begin{aligned} P(d) \quad &= \frac{1}{|A|}\sum_{a\in A}\mathbb{1}\left(\min_{b\in B}\|a-b\|<d\right) \\ R(d) \quad &= \frac{1}{|B|}\sum_{b\in B}\mathbb{1}\left(\min_{a\in A}\|b-a\|<d\right) \\ F\text{-}score(d) \quad &= \frac{2\cdot P(d)\cdot R(d)}{P(d)+R(d)} \end{aligned} \tag{10}$$

  where $\mathbb{1}(\cdot)$ is the indicator function, which returns 1 if the condition is true, and 0 otherwise.

- **Uniformity** quantifies how evenly points are distributed across a surface by measuring the local density variation. A lower score indicates a more uniform distribution, which reflects better surface coverage:

$$U(A,r) = \frac{1}{K}\sum_{i=1}^{K}\left(\frac{n_i-\bar{n}}{\bar{n}}\right)^2 \tag{11}$$

  where r denotes the ball radius (set to 0.05) for local density estimation and K represents random query points (set to 100) that are sampled from point cloud A.

- **Earth Mover's Distance (EMD)** measures the minimum amount of "work" required to transform one point cloud into another by moving points, where work is defined as the total distance points are moved. It provides a more geometrically faithful comparison than CD by considering the underlying distribution and preserving structural continuity:

$$\text{EMD}(A,B) = \min_{\phi:A\to B}\frac{1}{|A|}\sum_{a\in A}\|a-\phi(a)\|_2 \tag{12}$$

  where $\phi : A \to B$ is a bijection mapping each point in A to a unique point in B.

9. More About Results

Tab. 1 presents the averaged results of our fine-tuned models. Specifically, we performed an additional test/train split on the test set datasets of three scenarios for fine-tuning due to limited computational budget. Our BuildAnyPoint was fine-tuned on this sub-training set, using pairs consisting of diffusion model outputs and ground-truth meshes. Point2Building and City3D were also evaluated under the same subdivided setting. To fully demonstrate the effectiveness of fine-tuning, we additionally report experimental results without fine-tuning in Tab. 4, *i.e.*, where the models were not adapted to intermediate point cloud output and only trained on GT point-mesh pairs. The results in Tab. 4 are obtained using the full test set (around 10,000 for each scenario), in contrast to the subdivided set in Tab. 1 (around 3,000 instances in total). This distinction is driven by the fact that fine-tuning on the entire dataset would necessitate training on an additional 120,000 instances (around 40,000 for each scenario), a process that is computationally prohibitive in terms of time.

Table 4. **Quantitative comparison on structured mesh abstraction for all scenarios without fine-tuning.** Although the absence of a fine-tuning procedure leads to slightly over-tessellated outputs, our method still achieves the best performance in precision-related metrics.

	LiDAR Point Cloud					SfM Point Cloud					Sparse Sample Point Cloud				
Methods	# V ↓	# F ↓	# P ↓	FR ↓	CD ↓	# V ↓	# F ↓	# P ↓	FR ↓	CD ↓	# V ↓	# F ↓	# P ↓	FR ↓	CD ↓
City3D [21]	113	42	**14**	1 %	0.064	221	93	**16**	2 %	0.118	180	78	**13**	16 %	0.342
Point2Building [28]	**20**	**34**	18	1 %	0.053	**21**	**36**	18	2 %	0.053	**19**	**33**	16	1 %	0.044
Ours (w/o fine-tune)	42	77	28	**0 %**	**0.041**	37	68	23	**0 %**	**0.029**	37	69	24	**0 %**	**0.034**

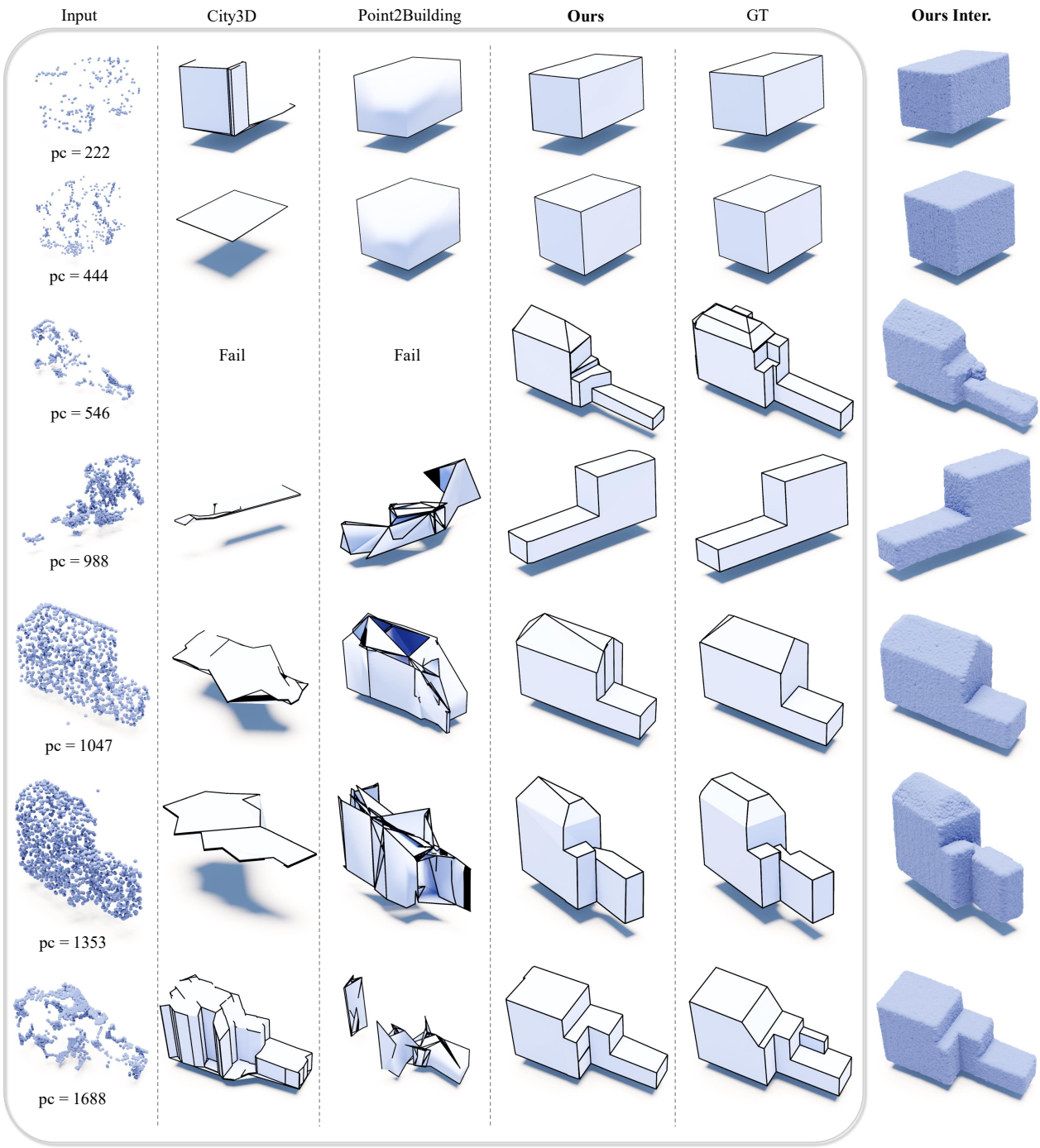

Figure 9. **More Results.** Our method consistently produces high-quality outputs, regardless of the input density.