Extend3D: Town-Scale 3D Generation

Supplementary Material

A. Appendix

A.1. Algorithms

Algorithm 1 Sparse Structure Generation

Input: $\mathcal{I}, \mathbb{P}$

$\boldsymbol{O}_0 \leftarrow \mathbf{1}_{\mathbb{P}}$
Define schedule
  $[t_{\text{start}} = t_1 > ... > t_k = 0]$
for $0 \leq n < n_{\text{iter}}$ **do**
  $\mathbf{Z}_0^{(g)} \leftarrow \mathcal{E}(\boldsymbol{O}_n)$
  Sample $\epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  $\mathbf{Z}_{t_1} \leftarrow (1 - t_{\text{noise}}) \cdot \mathbf{Z}_0^{(g)} + t_{\text{noise}} \cdot \epsilon$
  for $1 \leq m < k$ **do**
    Initialize $\hat{\boldsymbol{v}} \leftarrow \boldsymbol{v}(\mathbf{Z}_{t_m}, \mathcal{I}, t_m)$
    Optimize $\hat{\boldsymbol{v}}$ with Adam
    $\mathbf{Z}_{t_{m+1}} \leftarrow \mathbf{Z}_{t_m} + (t_{m+1} - t_m) \cdot \hat{\boldsymbol{v}}$
  end for
  $\boldsymbol{O}_{n+1} \leftarrow (\mathcal{D}(\mathbf{Z}_0) > 0)$
end for
return $\{\boldsymbol{p} : (\boldsymbol{O}_{n_{\text{iter}}})_{\boldsymbol{p}} > 0\}$

Algorithm 2 Structured Latent Generation

Input: $\{p_i\}, \mathcal{I}, \boldsymbol{P}$

Initialize $\mathbf{Z}_1$ with $\{\boldsymbol{p}_i\}$
Define schedule $[1 = t_1 > ... > t_k = 0]$
for $1 \leq m < k$ **do**
  Initialize $\hat{\boldsymbol{v}} \leftarrow \boldsymbol{v}(\mathbf{Z}_{t_m}, \mathcal{I}, t_m)$
  Optimize $\hat{\boldsymbol{v}}$ with Adam
  $\mathbf{Z}_{t_{m+1}} \leftarrow \mathbf{Z}_{t_m} + (t_{m+1} - t_m) \cdot \hat{\boldsymbol{v}}$
end for
return $\mathbf{Z}_0$

Algorithm 3 Extend3D

Input: $\mathcal{I}$

$\mathbb{P}, \boldsymbol{P} \leftarrow MoGe2(\mathcal{I})$
$\{\boldsymbol{p}_i\} \leftarrow SS(\mathcal{I}, \mathbb{P})$
$\mathbf{Z} \leftarrow \text{SLAT}(\{\boldsymbol{p}_i\}, \mathcal{I}, \boldsymbol{P})$
return $(\mathcal{D}_{\text{GS}}(\mathbf{Z}), \mathcal{D}_{\text{mesh}}(\mathbf{Z}))$

A.2. Image Patchification

We precisely patchify an input image using point cloud coordinates extracted from the monocular depth estimator. We can map a pixel $\mathcal{I}_{x,y}$ to a coordinate of the point cloud $q_{x,y}$ in the extended latent space. When $q_{x,y} \in \mathbb{W}_{i,j}$, the 3D area corresponding to the given pixel is in the patch (i, j) since the structure was initialized with the depth estimator. We therefore define the image patch $\psi_{i,j}(\mathcal{I})$ by collecting all $\mathcal{I}_{x,y}$ where corresponding $q_{x,y} \in \mathbb{W}_{i,j}$, setting the other pixels to be black, and cropping out black regions so that the image becomes square.

A.3. Dilated Sampling

We follow the recipe for dilated sampling in [21]. In this section, we assume that the unextended latent shape is $K \times K \times K$ for the sake of generalization. We divide the extended latent into $K \times K$ non-overlapping patches so that the size of each patch is $a \times b \times K$. We then randomly sample a pillar of 3D latent in each patch. The sampled pillars are attached by maintaining their relative positions to be a $K \times K \times K$ shaped latent. We sample $a \times b$ samples without replacement, and we call them dilated samples. The dilated samples are passed through the pre-trained model with the image condition $C_{\mathcal{I}}$ without image patchification. When dilated sampling is applied, Eq. (8) is altered to be:

$$\begin{aligned} \boldsymbol{v}(\mathbf{Z}_t, \mathcal{I}, t) = (1 - \gamma_t)\text{PatchWise}(\mathbf{Z}_t, C_{\mathcal{I}}, t) \\ + \gamma_t \text{Dilated}(\mathbf{Z}_t, C_{\mathcal{I}}, t), \end{aligned} \tag{14}$$

$$\gamma_t = 0.5 \cos^{\alpha}(\pi - \pi t) + 0.5, \tag{15}$$

where PatchWise is equal to Eq. (8) and α is a hyperparameter. We set $\alpha = 5$ in our experiments. We use dilated sampling only for sparse structure generation because we empirically observed that it worsens texture when applied to structured latent generation.

A.4. Computational Cost

Table 7. **Computational costs of 3D generation methods.** ($a = b = 2, d = 4, n_{\text{iter}} = 5$)

	VRAM (GB)	Time (min)
Trellis [44]	17	0.06
Hunyuan3D [47]	56	1.78
SynCity [7]	49	52.0
EvoScene [48]	68	35.3
Ours	**28**	**14.1**

We compared the computational cost of our method with previous methods in Tab. 7, using an NVIDIA RTX Pro 6000. Although our pipeline is heavier than object-centric methods (Trellis or Hunyuan3D), it requires substantially less memory and time than other scene-level pipelines.

Memory cost. For $a = b = 2$ and $d = 4$ case, the peak GPU memory used for our method is 28GB. Most of the memory is utilized in SLAT optimization process. Without SLAT optimization, it took 14GB VRAM. Also, for $a = b = 6$ and $d = 4$, without the SLAT optimization, the peak memory was 61 GB. Table 2 shows that our Extend3D achieves promising results even without SLAT optimization. Therefore, we can ensure reasonable quality with limited GPU resources.

Inference time. The computational complexity of our method is $O(abd^2 n_{\text{iter}})$. A larger (larger a and b), better detailed (larger d), and more complete (larger n_{iter}) scene requires more inference time, so there is a trade-off between output quality and inference time.

A.5. Rigorous Definition

Here, we provide a detailed and rigorous definition of $\phi^{\text{SS}}_{i,j}$, $\phi^{\text{SLAT}}_{i,j}$, $(\phi^{\text{SS}}_{i,j})^{-1}$, and $(\phi^{\text{SLAT}}_{i,j})^{-1}$ for clarity. Firstly, we define patchification $\phi_{i,j}$ through sliding windows as:

$$\mathbb{W}^K_{i,j} = \left[\frac{iK}{d}, \frac{iK}{d} + K\right) \times \left[\frac{jK}{d}, \frac{jK}{d} + K\right) \times [K], \quad (16)$$

$$\phi^{\text{SS}}_{i,j}(\mathbf{Z}^{\text{SS}}_t) = (\mathbf{Z}^{\text{SS}}_t)_{\mathbb{W}^N_{i,j}} \in \mathbb{R}^{N \times N \times N}, \quad (17)$$

$$\phi^{\text{SLAT}}_{i,j}(\mathbf{Z}^{\text{SLAT}}_t) = \left\{ \left(\boldsymbol{p} - \left(\tfrac{iM}{d}, \tfrac{jM}{d}, 0\right), \mathbf{z}_t\right) : (\boldsymbol{p}, \mathbf{z}_t) \in \mathbf{Z}^{\text{SLAT}}_t,\ \boldsymbol{p} \in \mathbb{W}^M_{i,j} \right\}. \quad (18)$$

The subtraction in Eq. (18) is for coordinate normalization into $[M]^3$. Secondly, the inversions of these processes are defined as:

$$(\phi^{\text{SS}}_{i,j})^{-1}(\boldsymbol{X})_{x,y,z} = \mathbf{1}_{(x,y,z) \in \mathbb{W}^N_{i,j}} \cdot \boldsymbol{X}_{x - \frac{iN}{d}, y - \frac{jN}{d}, z}, \quad (19)$$

$$(\phi^{\text{SLAT}}_{i,j})^{-1}(\boldsymbol{X}) = \left(\boldsymbol{X} + \left(\left(\tfrac{iM}{d}, \tfrac{jM}{d}, 0\right), \mathbf{0}\right)\right) \cup \left\{(\boldsymbol{p}, \mathbf{0}) : \boldsymbol{p} \in \{\boldsymbol{p}_i\},\ \forall \mathbf{z}\ (\boldsymbol{p}, \mathbf{z}) \notin \boldsymbol{X}\right\}, \quad (20)$$

setting the value of other positions to zero.

A.6. Ablation on Iterative SDEdit

We evaluated geometric results of different values of n_{iter}. The result with $n_{\text{iter}} = 0$ represents the decoded SLAT with geometry from the monocular depth estimator. Compared to the $n_{\text{iter}} = 0$ case, with a single under-noised SDEdit step, the geometry improved noticeably, indicating that our method can address incompleteness in depth estimation. This also implies that our method is robust to the minor errors of the depth estimator. While a single SDEdit step improved results, additional SDEdit steps sometimes degraded output quality, as shown in Tab. 8. Since the qualitative results, such as Figure 8, require iterations to complete the geometry, there is a trade-off between scene completion and geometric detail.

Table 8. **Ablation study on varying number of iterations n_{iter}.**

	CD ↓	F-score (0.05) ↑
$n_{\text{iter}} = 0$	0.0079	0.647
$n_{\text{iter}} = 1$	0.0075	0.681
$n_{\text{iter}} = 2$	0.0077	0.688
$n_{\text{iter}} = 3$	0.0082	0.677

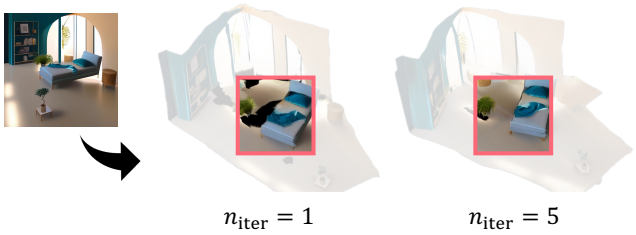

Figure 8. **Qualitative effect of iterative SDEdit**

A.7. More Results

We provide additional comparisons and large scene reconstruction results in Fig. 9, Fig. 10, Fig. 11, Fig. 12, Fig. 13, Fig. 14, Fig. 15, Fig. 16, Fig. 17, and Fig. 18.

A.8. Quantitative Evaluation Set

The evaluation sets used for appearance evaluation (LPIPS, SSIM, PSNR) and for geometry evaluation (CD, F-score) are illustrated in Fig. 19 and Fig. 20, respectively. We used the keywords from the original SynCity research for a text-prompted generation comparison. The keyword list is as follows: *autumn*, *campsite*, *campus*, *cyberpunk*, *mars*, *medieval market*, *medieval*, *oasis*, *paris*, *SF*, *solarpunk*, and *suburban*. Given a keyword, we used a prompt below for image generation before Extend3D:

> *Create a realistic, high-resolution aerial (bird's-eye) view of {keyword}. The image should be perfectly framed so that no buildings, objects, or landscape features are cropped or cut off. Show the entire scene clearly from directly above, with natural lighting, realistic depth, and detailed textures. The result should look like a professional drone photograph taken at high altitude, with balanced composition and clean edges.*

A.9. SLAT Decoding

While our main contribution is on the latent generation and most of our results in the paper are represented in 3D Gaussian Splatting, we also introduce an overlapping-patch SDF representation for mesh decoding. Unlike 3D Gaussians, if mesh patches are decoded disjointly and simply attached, the seams will be discontinuous, because there will be no faces. Therefore, similar to the overlapping patch-wise flow in our paper, we first decode SLAT into SDFs in an overlapping patch-by-patch manner. These SDF patches are then combined into a single SDF by averaging the values in the overlapping regions. Inspired by TRELLISWorld [2], we apply a cosine-based weighting to avoid near-zero SDF values at the seams. Finally, we extract the mesh from the combined SDF in a single step.

Unlike meshes, we decode SLAT into 3D Gaussians with disjoint SLAT patches, since it is not necessary to consider connectivity in 3D Gaussians.

Figure 9. **Example results for diverse images.** In this figure, we set $a = b = 2$. We compared our results with state-of-the-art open-source 3D generative models. The second image is from CarlaSC dataset [41]. The other image is generated by ChatGPT [30].

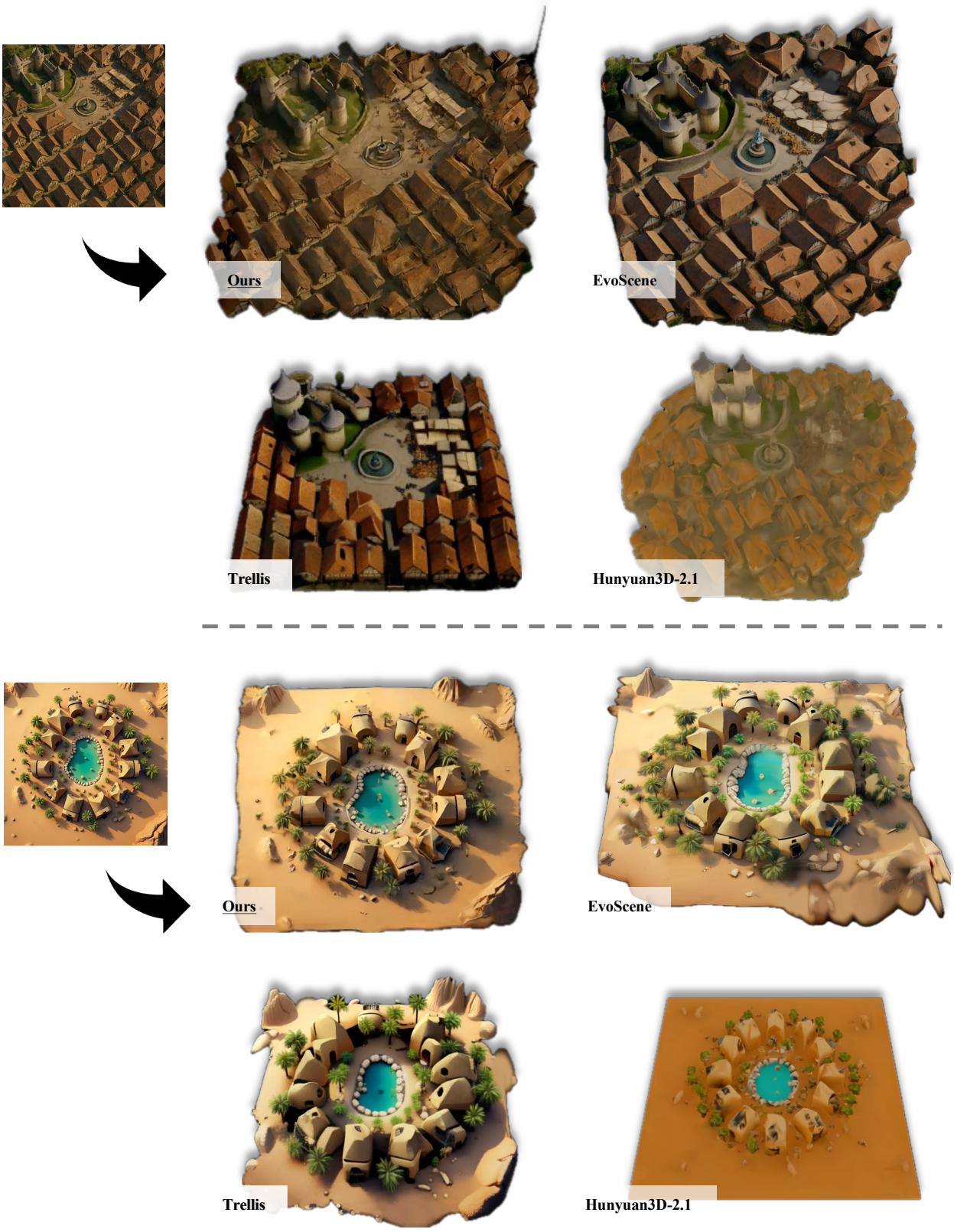

Figure 10. **Example results for diverse images.** In this figure, we set $a = b = 2$. We compared our results with state-of-the-art open-source 3D generative models. The images are generated by ChatGPT [30] and Flux.1 [dev] [15].

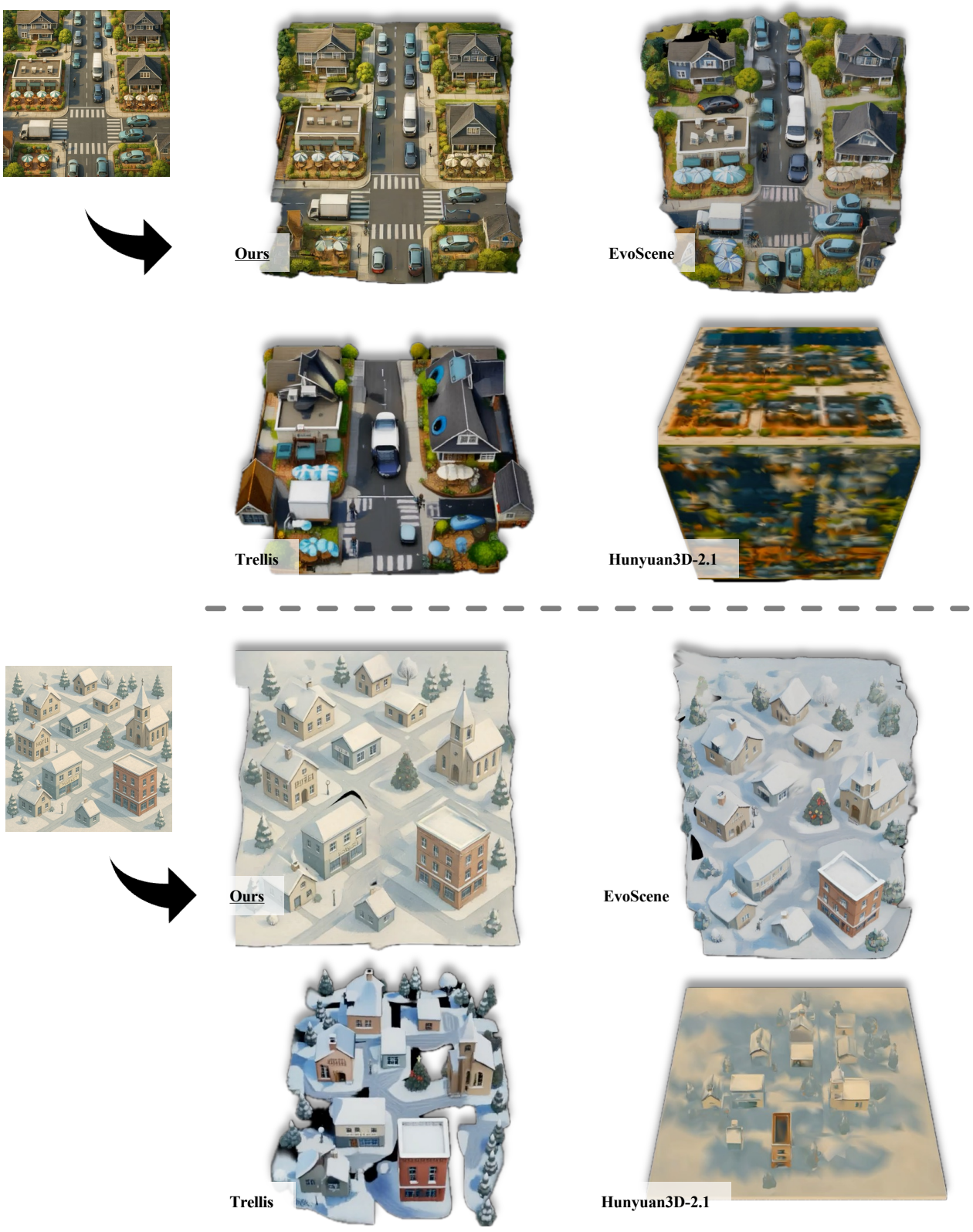

Figure 11. **Example results for diverse images.** In this figure, we set $a = b = 2$. We compared our results with state-of-the-art open-source 3D generative models. The images are generated by ChatGPT [30].

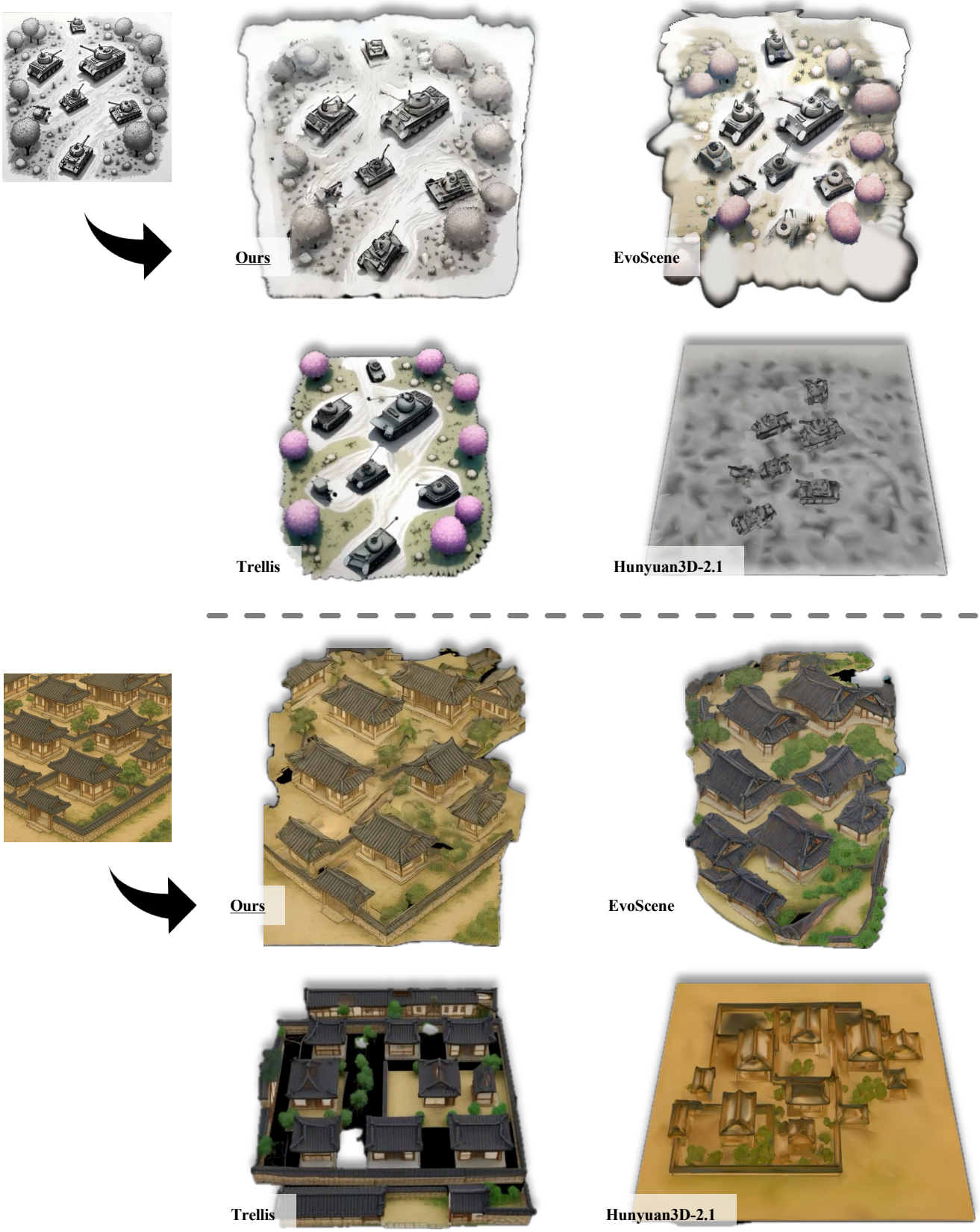

Figure 12. **Example results for diverse images.** In this figure, we set $a = b = 2$. We compared our results with state-of-the-art open-source 3D generative models. The images are generated by ChatGPT [30].

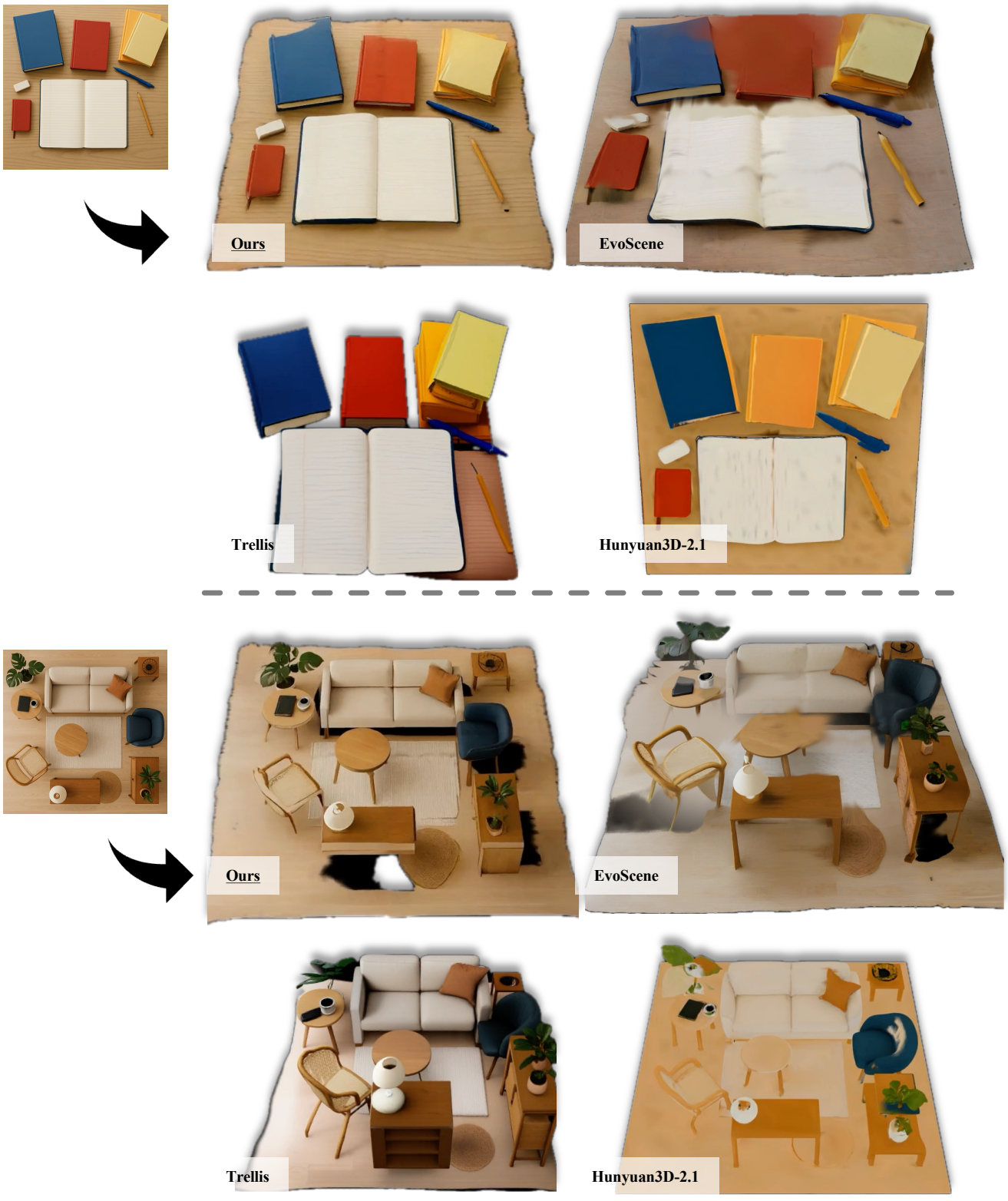

Figure 13. **Example results for diverse images.** In this figure, we set $a = b = 2$. We compared our results with state-of-the-art open-source 3D generative models. The images are generated by ChatGPT [30].

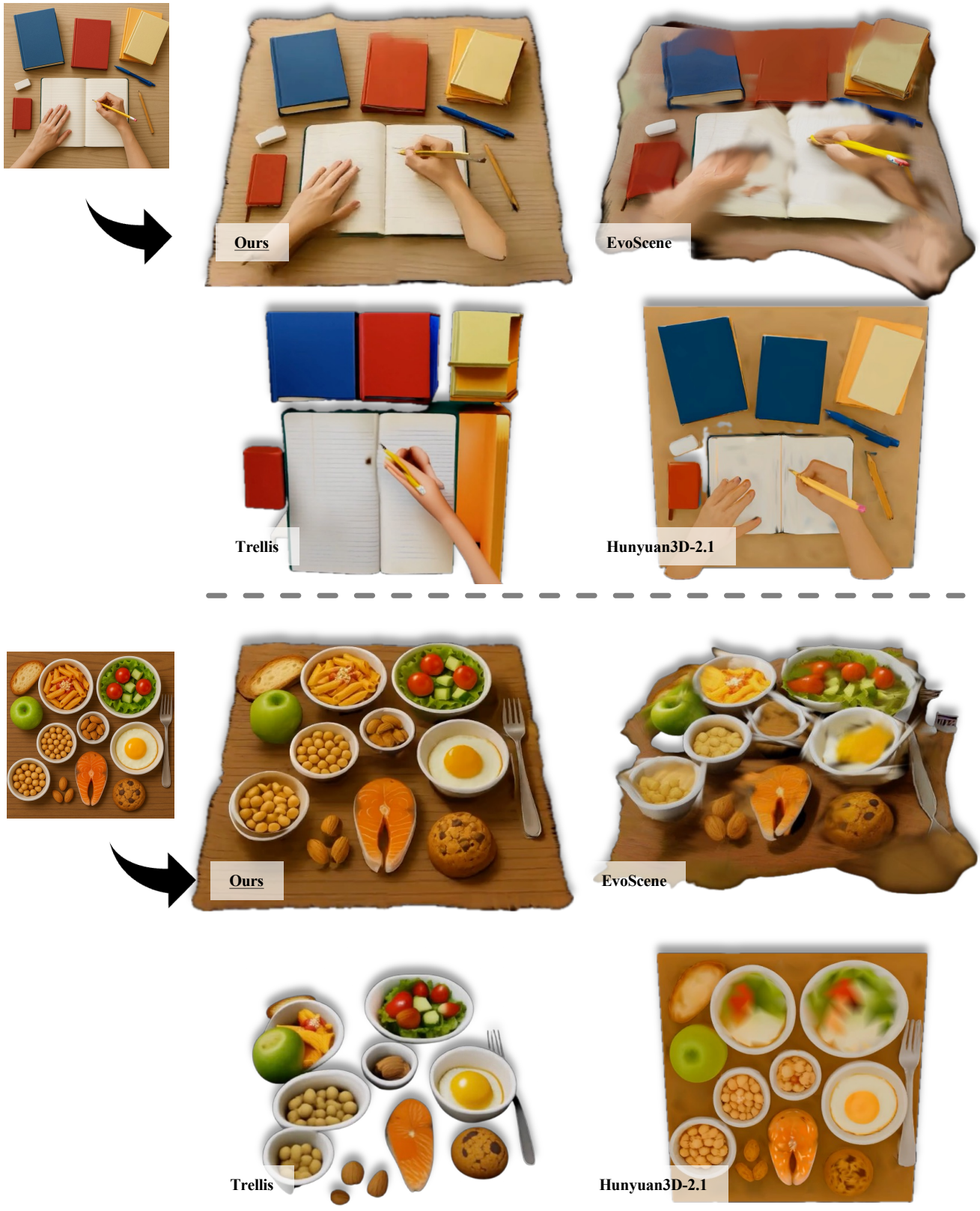

Figure 14. **Example results for diverse images.** In this figure, we set $a = b = 2$. We compared our results with state-of-the-art open-source 3D generative models. The images are generated by ChatGPT [30].

Figure 15. **Example results of comparison to SynCity.** We compared our results with SynCity [7]. Given a prompt, we first generated an image with ChatGPT [30] and used Extend3D for our results. The input prompts are on the left.

Ours EvoScene

Figure 16. **Example results of comparison to EvoScene [48].** In this figure, we set $a = b = 2$. The images are from ChatGPT [30] and UrbanScene3D [19].

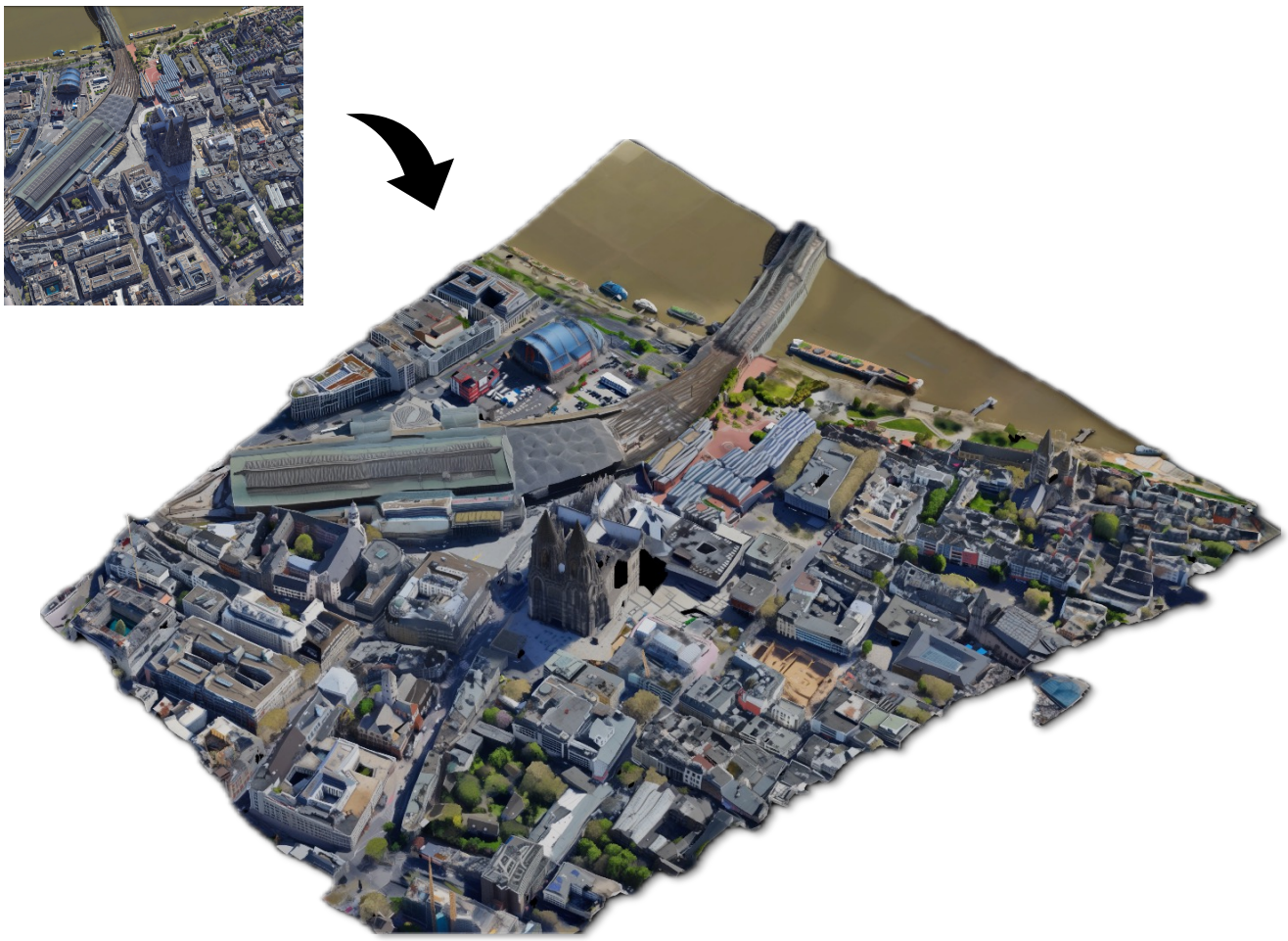

Figure 17. **The large scale result of Extend3D.** We generated large scale ($a = b = 6$) 3D scene from the image of Köln captured from Google Earth [9]. We didn't use SLAT optimization in this result due to the memory shortage.

Figure 18. **The large scale result of Extend3D.** We generated large scale ($a = b = 6$) 3D scene from the image of Athens captured from Google Earth [9]. We didn't use SLAT optimization in this result due to the memory shortage.

Figure 19. **100 images for appearance evaluation.** The images are from ChatGPT [30], Flux.1 [dev] [15], CarlaSC [41], Google Earth [9], and UrbanScene3D [19].

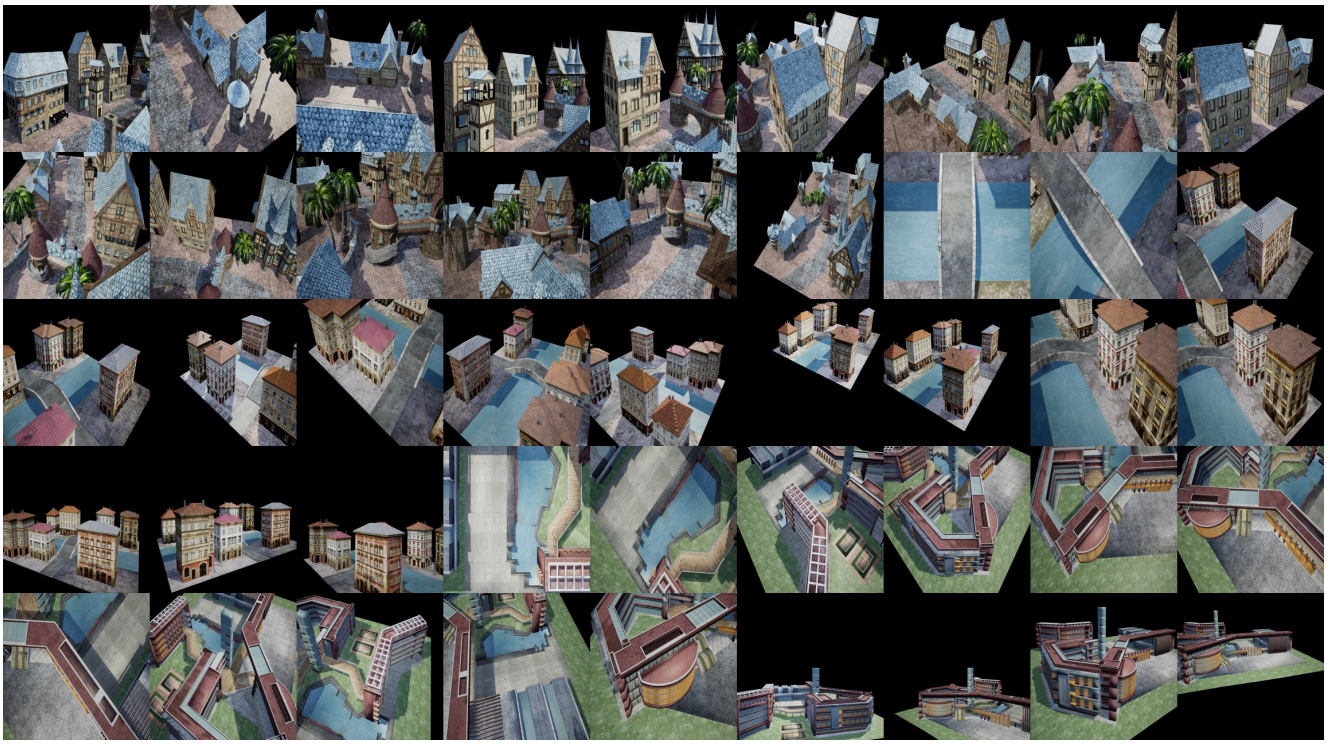

Figure 20. **45 images for geometry evaluation.** The images are from UrbanScene3D [19] synthetic scenes, together with ground truth meshes.