SynCity: Training-Free Generation of 3D Worlds

Paul Engstler* Aleksandar Shtedritski* Iro Laina Christian Rupprecht
Andrea Vedaldi
Visual Geometry Group, University of Oxford
{paule,suny,iro,chrisr,vedaldi}@robots.ox.ac.uk

Figure 1. We introduce **SynCity**, a novel method for generating complex and freely navigable 3D worlds from a prompt. It is training-free and leverages powerful language, 2D, and 3D generators through new prompt engineering strategies.

Abstract

We propose SynCity, a method for generating explorable 3D worlds from textual descriptions. Our approach leverages pre-trained textual, image, and 3D generators without requiring fine-tuning or inference-time optimization. While most 3D generators are object-centric and unable to create large-scale worlds, we demonstrate how 2D and 3D generators can be combined to produce ever-expanding scenes. The world is generated tile by tile, with each new tile created within its context and seamlessly integrated into the scene. SynCity enables fine-grained control over the appearance and layout of the generated worlds, which are both detailed and diverse.

*Denotes equal contribution. Project page: https://research.paulengstler.com/syncity/

1. Introduction

We consider the problem of generating 3D worlds from textual descriptions. Generating 3D content, for example, for video games, virtual reality, and simulation, is highly laborious and time-consuming. This is particularly true for large 3D scenes, even though these are often in the background and may have limited artistic significance. Automating their generation is thus particularly appealing.

The advent of modern generative AI has already impacted 3D generation, and in particular, the generation of 3D objects. DreamFusion [40] was among the first to adapt diffusion-based 2D image generators [44] to create 3D objects. Subsequent advancements fine-tuned 2D image generators to produce multiple consistent views of an object [14, 33, 48, 53] and learned few-view 3D reconstruc-

tion networks [25, 65]. More recently, the focus has shifted to methods that learn 3D latent spaces [10, 26, 27, 63, 69]. 3D latents can be sampled to generate 3D objects directly and with better geometry. Furthermore, by making these 3D latent generators conditional on an image prompt, they can be easily combined with 2D image generators, which can generally be trained on a much larger scale.

In addition to objects, there is also ample literature on generating 3D scenes. Most scene generators are image-based and progressively reconstruct scenes by expanding from an initial image [8, 12, 13, 17, 31, 38, 43, 67, 70], combining depth prediction, image and depth outpainting, and 3D reconstruction using NeRF [36] or 3D Gaussian Splatting [20]. The main advantage of these approaches is that they also leverage powerful 2D image generators to create the various views of the scene. These 2D generators understand complex textual prompts and result in vibrant 3D scenes with good artistic quality. However, it is challenging to keep the scene consistent while expanding it. As a consequence, while the reconstructed scenes can envelop the observer in a 360° manner, it is often not possible to 'walk' into the scene for more than a few steps [22].

A key challenge in generating scenes beyond these '3D bubbles' is maintaining consistency incrementally without drifting. We argue that latent-space 3D generators might help, as they can regularize and constrain the reconstructed geometry, including hallucinating shapes and textures in regions *behind* the visible sides of objects. Some evidence comes from recent works like BlockDiffusion [60] and LT3SD [34], which learn to generate large coherent 3D scenes in latent space. However, they do not leverage 2D generators trained on billions of images, which severely hinders scene diversity and limits their ability to generalize and follow instructions.

In this work, we aim to combine the strengths of latent 3D and 2D generators to create large, high-quality 3D scenes that can be navigated freely (Fig. 1). First, we note that, while 3D generators like TRELLIS [63] are trained for object-level reconstruction, they can reconstruct fairly complex compositions of multiple objects. Borrowing ideas from video game world building, we show that TRELLIS can effectively generate, if not an entire world, at least *tiles* representing local regions of the world. In particular, we show that, if we prompt the model with an 'isometric' view of a tile, we can effectively generate it in 3D.

Given this basic ability, we then address the problem of generating the images of the tiles that form a larger scene. To do so, we build on a text-to-image generator (Flux [21]) and introduce a new way of prompting it that reliably causes it to output tile-like images, with a consistent isometric framing. This regular and stable framing is the key reason why the reconstructed 3D tiles can fit together seamlessly.

In addition to framing, we propose two additional mechanisms to make tiles fit together well. First, we encourage consistency in appearance by using previously generated tiles to provide context for the image generator, where each new tile *inpaints* a missing region in a 2D isometric view of the scene. Second, we enforce geometric consistency by blending the 3D representations of neighboring tiles using the 3D generative model.

Tile-specific textual prompts control the generation of the tiles. Tile prompts can also be generated automatically, utilizing a large language model (ChatGPT [37]), so that an entire scene can be obtained from a single 'world' prompt.

As we show in the experiments, SynCity can, in this way, leverage off-the-shelf components (*i.e.*, language, 2D and 3D generators) to produce vibrant, detailed, and coherent 3D worlds that can be navigated freely.

2. Related Work

Novel view synthesis for scenes. Expanding an image beyond its boundaries has been a long-standing task in computer vision. Early methods that sought to expand object-centric scenes relied on layer-structured representations [24, 35, 49, 51, 55, 56], which disregard the scene's actual geometry. SynSin [59] is a pivotal work where image features are projected and used as conditioning to generate novel views, achieving geometric and semantic consistency. ZeroNVS [45] introduces high-quality results with fine-grained control of the camera but remains object-centric. GenWarp [46] integrates semantic information through cross-attention when generating a novel view.

A challenge for these methods is to avoid semantic drift and maintain object persistence. To obtain a 3D representation, the generated views need to be transferred into such a representation, *e.g.*, NeRF [36] or Gaussians [18, 20], where any geometric conflicts must be resolved.

Image projection-based scene generation. A different line of work follows the paradigm of building the 3D representation of a scene sequentially using 2D image generators [8, 13, 17, 23, 38, 43, 58, 61, 67, 70]. Most of these methods employ an image generator to outpaint the existing scene using predefined camera poses. The results are then fused in 3D with depth prediction models. Text2Room [17] generates meshes of indoor scenes. As the bounds of the mesh delimit the scene, it can be freely explored. LucidDreamer [8] and Text2Immersion [38] go beyond indoor scenes, but their generated scenes reveal geometric inconsistencies when stepping away from the camera poses used to generate the scene. Invisible Stitch [12] addresses this issue by inpainting depth (rather than naively aligning it), and RealmDreamer [50] proposes multiple optimization losses to refine the generated scene. Despite these improvements, the resulting scenes still suffer from geometric artifacts and remain relatively small. WonderJourney [67] introduces

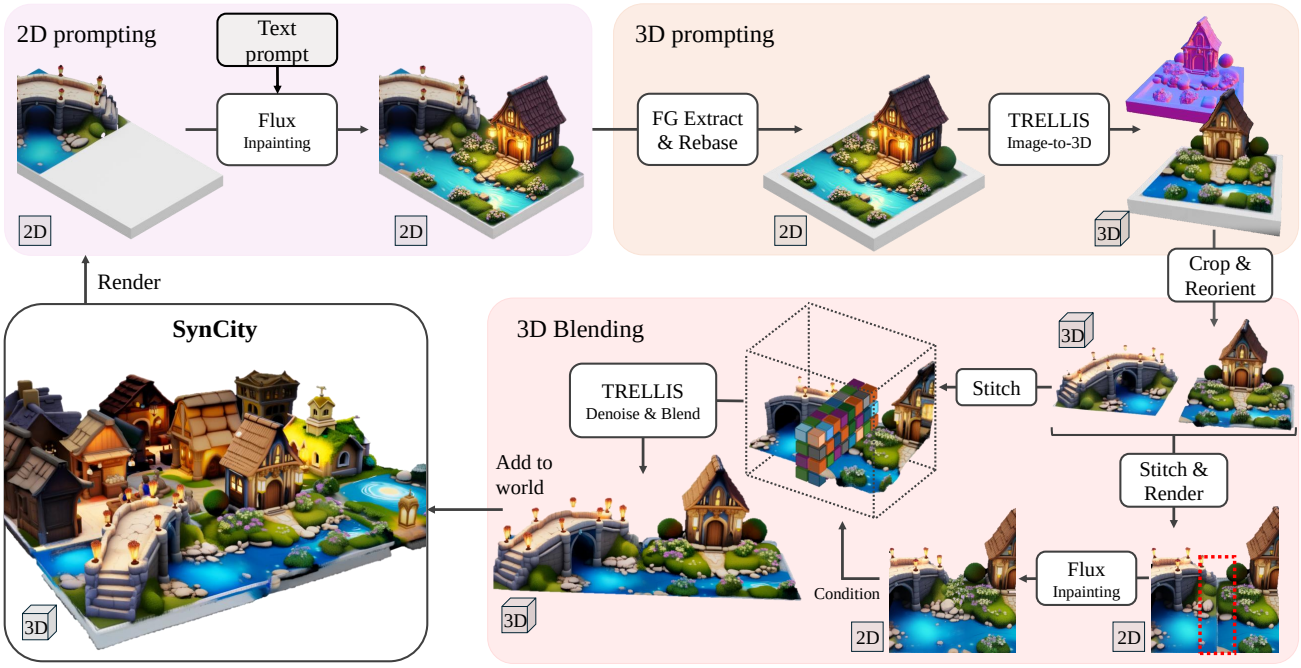

Figure 2. **Overview of SynCity.** *2D prompting:* To generate a new tile, we first render a view of where the tile should be placed, including context from neighboring tiles. *3D prompting:* We extract the new tile image and construct an image prompt for TRELLIS by adding a wider base under the tile. *3D blending:* The 3D model output by TRELLIS is often not well blended with the rest of the scene. To address this, we render a view of the new tile alongside each neighboring tile and inpaint the region between them using an image inpainting model. Next, we condition on this well-blended view to refine the region between the two 3D tiles. Finally, the new tile is added to the world.

novel ideas for depth fusion, such as grouping objects at similar disparity to planes and refining sky depth, enabling extensive 'scene journeys' where independent representations are built between scene 'keyframes'. However, these are not merged into one coherent scene. WonderWorld [68] leverages these improvements to build a single scene, allowing interactive updates, but the true extent of the generated scenes remains limited. Other works use panoramas [52, 57] or implicit representations [2, 45], but the freedom of movement remains constricted.

Procedural scene generation. Further methods permit long-range fly-overs over nature [5–7, 28, 31] or cities [30, 47, 64]. These methods usually generate procedural unbounded images (*e.g.*, the terrain makeup or a city layout). While these methods create realistic-looking images, they are often monotonous as they are domain-specific and thus highly constrained in the variety they can generate.

3D scene generation. Instead of merely generating images of a scene or outpainting it only in 2D, other methods generate the 3D representation directly. Set-the-Scene [9] adds a layer of control to the layout of NeRF scenes by defining object proxies. BlockFusion [60] learns a network to diffuse small blocks to extend a mesh auto-regressively. A 2D layout conditioning is used to control the generation process, allowing users to generate scenes of rooms, a village, and a city. While the method allows building large-scale scenes, the variety of the objects it generates is severely limited as it requires domain-specific 3D training data. Furthermore, it generates untextured meshes. LT3SD [34] learns a diffusion model that generates 3D environments in a patch-by-patch and coarse-to-fine fashion. However, this method is only trained to produce indoor scenes.

At the same time, the synthesis of complex, high-fidelity objects has been enabled by the rapid progress in the fields of text-to-3D and image-to-3D generation [26, 27, 29, 32, 41, 42, 48, 54, 63, 71, 73]. Trained on large-scale curated subsets of 3D datasets such as Objaverse-XL [11], these models can generate a large variety of different objects.

While prior works have utilized 3D object generators to composite a scene from objects [19, 66], we generate complete chunks of a scene that are seamlessly fused together, which is a fundamentally new approach to scene generation.

3. Method

Our goal is to generate a 3D world $\mathcal{G}$ from an initial textual prompt p_0. Our method, SynCity, leverages prompt engineering in combination with *off-the-shelf* language, 2D, and

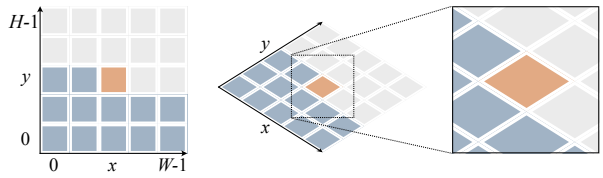

Figure 3. *Left:* Progressive generation of world tiles $\mathcal{T}$. *Right:* Isometric framing of a tile for image-based prompting.

3D generators to create the entire world automatically, with no need to retrain the models.

We structure the world as a $W \times H$ grid $\mathcal{T} = \{0, \dots, W-1\} \times \{0, \dots, H-1\}$ of square tiles, each of which can contain several complex 3D objects (*e.g.*, buildings, bridges, trees) as well as the ground surface. We generate the world progressively, tile by tile, as shown in Fig. 3. When generating tile $(x, y) \in \mathcal{T}$, tiles $\mathcal{T}(x, y) = \{(x', y') \in \mathcal{T} : y' < y \vee (y' = y \wedge x' < x)\}$ have already been generated.

An overview of our approach is shown in Fig. 2. The first step is to expand the world description p_0 into tile-specific prompts (Sec. 3.1). The second step is to pass these tile-specific prompts to a 2D image generator and inpainter to create an isometric view of each tile, accounting for the part of the world generated so far (Sec. 3.2). The third step is to extract image prompts from these isometric views and use them as input to an image-to-3D generator to reconstruct each tile's geometry and appearance in 3D (Sec. 3.3). The final step is to align and blend the 3D reconstructions of the tiles to create a coherent 3D world (Sec. 3.4).

3.1. Prompting the Language Model

The goal of language prompting is to take a high-level textual description of the world p_0 and expand it into a set p of tile-specific textual prompts that can be used to generate the 3D world. Specifically, p is a collection of sub-prompts $p_{xy} \in \Sigma^*$, one for each tile, and a world-level 'style' prompt $p_\star \in \Sigma^*$, so that we can write $p = \{p_{xy}\}_{(x,y)\in\mathcal{T}} \cup \{p_\star\}$, where Σ^* is the set of all possible strings.

The prompt p can be constructed manually (allowing control over the content of each tile) or generated by a large language model (LLM) such as ChatGPT [37] from a 'seed' prompt. For the latter, we employ in-context learning, prompting ChatGPT o3-mini-high to generate a grid-like world with tile-specific descriptions after providing it with a template in JSON format (see Appendix A.1).

3.2. Prompting the 2D Generator

We use the language prompts p from Sec. 3.1 to prompt an off-the-shelf 2D image generator Φ_{2D} to output a 2D image $I(x, y)$ of each tile to be generated, as shown in Fig. 4. The image $I(x, y)$ must satisfy several constraints: (1) It must reflect the tile-specific prompt p_{xy} of the target tile as well as the world-level prompt $p_\star$. (2) It must be suitable for prompting the image-to-3D generator in the next step. (3) It must be consistent with the previously generated tiles.

We propose a prompting strategy designed to satisfy these constraints. The image is drawn as a sample $I(x, y) \sim \Phi_{\text{2D}}(q, B, M)$ from the 2D image generator Φ_{2D}, where $q = p_{xy} \cdot p_\star$ is a prompt that combines the tile-specific and world-level descriptions. The generator Φ_{2D} is also provided with a base image B and an inpainting mask M to constrain its output, es explained next.

Tile inpainting. To satisfy constraint (2), we encourage the image generator to produce regular tiles so that the image-to-3D model outputs tiles with regular geometry that fit well together. We assume that tiles have a square basis of unit size and are imaged in an 'isometric' manner. This framing of the tiles is conducive to generating regular 3D tiles. Furthermore, it is a common choice in video games and might have been observed by the image generator during training, as these models are often trained on game-like data.

While we could fine-tune the image generator Φ_{2D} to produce such images, we demonstrate below that this effect can be achieved through prompt engineering alone, avoiding the need for retraining. We only assume that Φ_{2D} is capable of inpainting — a common feature of modern image generators, and provide it with inputs B and M, as shown in Fig. 4. Specifically, B is set to be the image of the base of the tile, represented as a square, gray slab imaged from a fixed isometric vantage point. The mask M is a binary mask covering a cube on top of the base.

Figure 4 shows the result of prompting the model in this manner, as well as what happens if signals B and M are removed: the viewpoint and general frame of the tile become random and unsuitable for 3D generation.

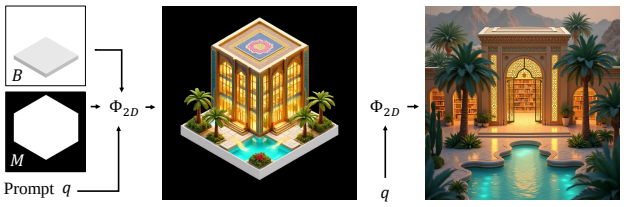

Figure 4. *Left:* Generation of the 2D image prompt for the first world tile at $x = 0$ and $y = 0$. The image generator Φ_{2D} is conditioned on $q = p_{00} \cdot p_\star$ and tasked with inpainting the base image B within the masked region M. *Right:* Without 'framing' the image using B and M, the output image is unsuitable for tiling.

Context-aware generation. Except for the first tile $(0, 0)$, tiles are generated in the context of the part of the world generated before. To account for this context, for tiles with $x, y > 0$, we modify the base image B and the mask M as shown in Fig. 5. For the base image B, instead of the slab, we render the previously generated portion of the 3D

Figure 5. *Left:* Base image B and inpainting mask M (white overlay) used to prompt the image generator Φ_{2D} to generate an image for a world tile at $x > 0, y > 0$. *Right:* Result of inpainting.

world (which is constructed as described next in Secs. 3.3 and 3.4), providing context to the inpainting network. We also modify the mask M to avoid covering tiles already generated to the left (west), *i.e.*, for a tile $(i, j) \in \mathcal{T}$, these are tiles $\{(x, y) \in \mathcal{T} : x < i \wedge y = j\}$. See the appendix for a comparison of different masking schemes.

To ensure continuity of the ground, before rendering the contextual image, we trim any 3D geometry that is sufficiently high to occlude the tile we wish to generate, as shown in Fig. 6 (observe the trimmed structures in Fig. 5).

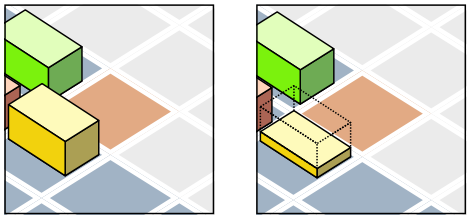

Figure 6. Trimming tall structures for 2D prompting

The appendix discusses a special case for tiles at the boundaries of the world (see Appendix A.2).

3.3. Prompting the 3D Generator

Given the tile image $I(x, y)$ obtained from the 2D image generator in Sec. 3.2, the next goal is to generate a corresponding 3D reconstruction $G(x, y)$ of the tile using an image-to-3D model Φ_{3D}. We opt for a robust 3D generator and select TRELLIS [63] due to its strong performance, ability to generate both shape and texture, and latent space structure, which is easy to manipulate for blending, as we show later in Sec. 3.4. TRELLIS produces the reconstructions $G(x, y)$ in the form of 3D Gaussian Splats (3DGS).

Thus, 3D reconstruction amounts to drawing a sample $G(x, y) \sim \Phi_{\text{3D}}(J(x, y))$ from the image-to-3D generator Φ_{3D}. Rather than conditioning on the image $I(x, y)$, we use a pre-processed version $J(x, y)$, as described next.

2D foreground extraction and rebasing. Recall that the image $I(x, y)$ output by the 2D generator in Sec. 3.2 is an image of the tile and its context. However, the 3D generator

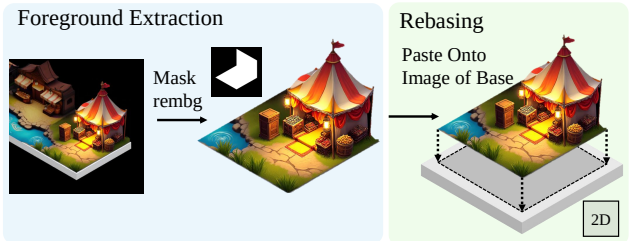

Figure 7. *Left:* Isolating the image of the new tile from $I(x, y)$. *Right:* Adding a slightly larger base underneath the tile.

Φ_{3D} expects the input image to focus only on the object that needs to be reconstructed, *i.e.*, the new tile. The first step is to extract from $I(x, y)$ only the part that corresponds to the new tile, which we achieve using `rembg` [15] with alpha matting [4], as shown in Fig. 7.

The resulting image is narrowly cropped around the new tile. Similar to Sec. 3.2, we found it beneficial to include a slab base for the tile, an operation we call 'rebasing,' as shown in Fig. 7. We simply compose the image of the tile with a slightly larger gray slab (in 2D) to obtain $J(x, y)$, which effectively provides a 'frame' for the 3D generator to work with. The base is reconstructed as part of the tile's geometry, which can be used for validation and as an easy-to-detect handle for further 3D processing.

The 'rebased' image $J(x, y)$ is fed to the 3D generator Φ_{3D} to obtain the 3D reconstruction $G(x, y)$ of the tile, in the form of 3D Gaussian Splats. The effect of rebasing on the 3D result is shown in Fig. 8.

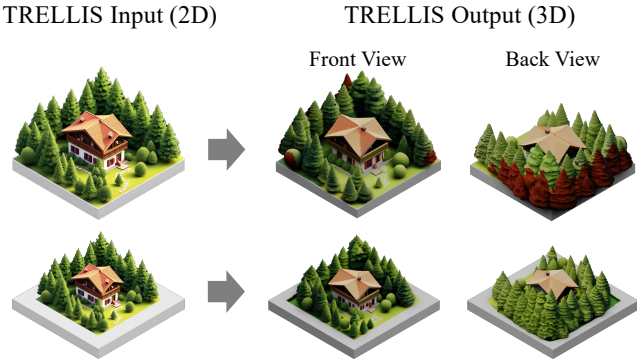

Figure 8. *Top:* 3D reconstruction using a tight base. *Bottom:* 3D reconstruction with a slightly larger base, which helps to keep the tile's geometry above ground (see the back of the reconstruction) and creates an easy-to-detect 3D base.

3D geometric validation. Because the generators are imperfect, we verify the 3D reconstruction $G(x, y)$ to ensure that it is of sufficient quality. If not, we discard it and regenerate the tile using a different random seed. To verify the tile, we use a few heuristics to check that the tile's ge-

ometry occupies a square region of sufficient size and that the base of the tile has been reconstructed faithfully. Please see Appendix A.3 for more details.

3D post-processing. At this point, we have verified the 3D reconstruction $G(x, y)$ for the tile as a mixture of 3D Gaussians. However, the actual 3D footprint, orientation, and size of the tile are controlled by the 3D generator and are inconsistent. The post-processing step utilizes simple heuristics to crop the subset of 3D Gaussians that form the actual tile, remove the extended base, rescale them to fill the full tile, and reorient them to face the same way as the 2D image prompt. We explain this in more detail in Appendix A.4.

3.4. 3D Blending

At this stage of the pipeline, we have reconstructed all 3D tiles $G(x, y)$, $(x, y) \in \mathcal{T}$. Due to the prompting and post-processing steps in Secs. 3.2 and 3.3, the tiles are already approximately aligned and oriented correctly, including being roughly level with the ground.

However, they are not perfectly aligned, particularly at their boundaries. This misalignment arises because TRELLIS does not reconstruct the input images exactly, and only a single view of each tile is provided, which only indirectly controls the reconstruction of the back of the tile. Further, while $G(x, y)$ is represented as 3DGS, these imperfections are not easily addressed in that representation space.

We address this issue by blending the tiles in TRELLIS latent space to create a coherent and continuous 3D scene. As explained next, we first repaint the boundary region in 2D. Then, we align the tiles in the latent voxel grid of Φ_{3D}. Finally, we resample the voxel features in a narrow boundary region between two tiles.

Blending in 2D. To blend the latents of two neighboring tiles, we first predict the appearance of the boundary between the two tiles. To achieve this, we place the two 3D tiles next to each other, render a frontal view, and inpaint the middle region of the rendering (Fig. 2) using Φ_{2D}. This results in a blended image, which we use to condition Φ_{3D}.

Unifying the tile size in 3D latent space. Recall that, due to the rebasing, $G(x, y)$ contains a 3D base. While we have removed the base in 3DGS space, we have yet to do the same in the latent space. We use the same cuts applied in 3DGS space to crop the latents, rounding them to account for the discrete nature of the latent voxel grid.

Because these cuts might differ for two neighboring tiles, γ^1 and γ^2, we may need to upsample the latents to ensure $\gamma^1, \gamma^2 \in \mathbb{R}^{D\times R\times R\times R}$. Here, γ represents D-dimensional features in the R-sized 3D grid that TRELLIS denoises.

We found that naïvely upsampling the latents by interpolation leads to poor reconstructions, as shown in Fig. 9. Instead, we propose a different scheme where we resample the features (called *structured latents* in [62]) after upsampling the latent voxel grid of each tile. First, we upsample the cropped occupancy volume that TRELLIS predicted to the original resolution $V \in \{0, 1\}^{R\times R\times R}$. Next, we denoise *a new set* of latents γ on the upsampled occupancy volume. To preserve the details and textures of the original 3D tile, we render it from multiple views and jointly condition the structured latent inference on all of them. In practice, when denoising with multiple conditioning views, at each timestep, the denoising step is computed as the average denoising step across all views. We show that this upsampling scheme leads to superior reconstructions in Fig. 9.

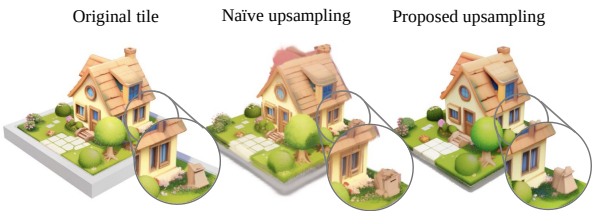

Figure 9. **Upsampling sparse latents.** We need to resize or upsample sparse latents in order to stitch them. Due to the sparsity of the latents and the behaviour of the latent decoder, naively resampling in latent space leads to artifacts. Our proposed resizing of the sparse latents better preserves textures and fine structures.

Now, mirroring the base cropping operation in 3DGS space, we have tiles of matching sizes in latent space.

Blending in 3D. Finally, we use Φ_{3D} to blend tiles. We take the latents of the two tiles γ^1 and γ^2, where $\gamma^1, \gamma^2 \in \mathbb{R}^{D\times R\times R\times R}$ after upsampling. We combine them into a new volume γ, where the side where they meet is in the center:

$$\gamma_{:,x,y,z} = \begin{cases} \gamma^1_{:,x+R/2,y,z}, & \text{if } x < R/2, \\ \gamma^2_{:,x-R/2,y,z}, & \text{if } x \geq R/2. \end{cases}$$

We apply the denoising function Ω, which is the latent denoiser of Φ_{3D}, to the volume γ, but only within the middle region where we have applied the stitch, *i.e.*, for $x \in [R/2 - r, R/2 + r]$ for some $r < R/2$, while keeping the rest fixed. Formally, we initialize $\tilde{\gamma} \sim \mathcal{N}(0, I)$ and at each denoising step t, we update $\tilde{\gamma}$ as:

$$\tilde{\gamma}_{t+1,:,x,y,z} = \begin{cases} \Omega(\tilde{\gamma}_{t,:,x,y,z}), & \text{if } |x - R/2| \leq r, \\ \gamma_{t+1,:,x,y,z}, & \text{otherwise.} \end{cases}$$

Here, γ_t is obtained by adding noise to the original γ at the corresponding noise level for step t. In practice, we only update the structured latents, keeping the sparse structure (latent voxel grid) fixed: The low spatial resolution of the sparse structure ($R = 16$, compared to $R = 64$ for the structured latents) is too coarse for choosing an adequate r.

Win Rate (%)				
Overall	Geometry	Exploration	Diversity	Realism
90.9	81.8	90.9	90.9	86.4

Table 1. Win rates of our method against BlockFusion. We asked participants to select which scene they prefer overall, as well as which one has better geometry, would be more interesting to explore, is more diverse, and has better realism.

Figure 10. *Left:* 2×2 grid generated with our method, *not* taking context into account — here, the scale of the buildings is not consistent. *Right:* Generated with our method using the same prompts, where context is taken into account as described in Sec. 3.2.

4. Experiments

Experimental details. We generate the text prompts using ChatGPT o3-mini-high. For the 2D inpainter, we use the Flux ControlNet of [1].

Human preference. We evaluate human preference for the results generated by our method compared to those obtained with BlockFusion [60]. In particular, we compare a 'city' scene, showing the entire scene as well as close-up detail views. As seen in Tab. 1, participants ($n = 22$) find our method better overall, with superior geometry, realism, and diversity.

4.1. Ablations

Here, we ablate several components of our approach to demonstrate the importance of each.

Building a grid. A naive approach to generating a 3D scene involves querying the image generator to produce an image of a large-scale scene (using our 2D image prompt setup) and then obtaining the entire 3D world *directly* with TRELLIS. To achieve the same level of control provided by our method, the textual prompt needs to be highly detailed and include layout instructions. However, we found that neither precise nor abstract prompts were effective at steering the generations of Flux (for details, see Appendix A.4). This highlights the effectiveness of our grid-based approach in generating highly detailed 3D worlds at scale.

2D prompting context. We remove the context from neighboring tiles, as described in Sec. 3.2. When this is done, each tile is sampled independently, and the relative scale between objects becomes inconsistent (Fig. 10).

Method	Base Area	Squareness ↑	Completeness ↑
No Rebasing	2271	0.92	0.73
Ours	4096	**1.00**	**1.00**

Table 2. Average tile 3D geometry metrics for an approach without rebasing and our method. Rebasing is crucial to ensure a tile is square and its base has been reconstructed faithfully. The metrics we use are the area of the base in voxels, a measure for the 'squareness' of the base, and how many border voxels have been faithfully reconstructed. For details, please refer to the appendix.

Method	LPIPS ↓	SSIM ↑	FID ↓	KID ↓
Naïve upsampling	0.5914	0.3093	200.5	0.243
Ours (single frame)	0.3517	0.5149	111.6	0.069
Ours (multi frame)	**0.3212**	**0.5312**	**89.1**	**0.051**

Table 3. Perceptual metrics for our method and the naive approach for 3D latent upsampling. Lower values for LPIPS [72], FID [16], and KID [3] are better, while higher values for SSIM are better. We see that even using a single conditioning frame leads to better results, and multiple frames further improve performance.

Rebasing. To place tiles on a grid, they need to be square (otherwise the grid would be jagged), and their base must be reconstructed faithfully (clearly delimiting where the tile stops). Without rebasing, the geometry generated by TRELLIS might extend beyond the base, making the tile's 'true' extent difficult to detect, as shown in Fig. 8. We ablate the effect of rebasing using a small 2×2 scene to minimize the effect of error accumulation. As seen in Tab. 2, without rebasing, TRELLIS generates tiles that are, on average, neither perfectly square nor have a solid border.

Latent upsampling. We sample 10 random views from each of 200 tiles generated by TRELLIS and compute perceptual metrics in Tab. 3 when upsampling latents with our proposed approach in Sec. 3.4 versus naive interpolation. Our proposed method leads to better results across a range of metrics, even when using a single conditioning frame.

3D blending. In Fig. 12, we generate a scene without applying 3D blending (Sec. 3.4), resulting in visible discontinuities between the tiles.

4.2. Qualitative Results

We present example scenes generated by our method in Fig. 1. Additionally, we show detailed views highlighting the quality and diversity of the scenes. Please see the appendix for many more examples.

Exploring a generated world. SynCity produces explorable worlds that are easy to navigate. To demonstrate this, we sample trajectories and 'walk into' the generated 3D worlds (Fig. 11). Pre-made skybox textures are added

Figure 11. **Exploring a 3D world.** We show camera walk-throughs that explore the generated 3D worlds. Please refer to the supplement for accompanying videos.

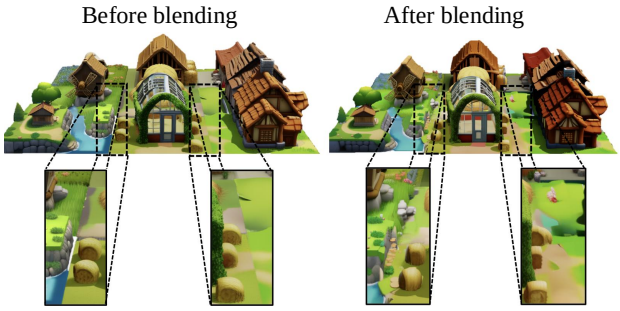

Figure 12. *Left:* Tiles before the 3D blending step. *Right:* After the 3D blending step. Previously visible boundaries between tiles are now well-blended, resulting in a more coherent appearance.

for visual effect. Unlike videos generated by world models such as [39], our results are guaranteed to be consistent and do not suffer from semantic drift. Unlike other systems that only generate a 'bubble,' our method creates spaces sufficiently large to be navigated in a meaningful way.

5. Conclusion

We have introduced SynCity, an approach to generate complex, diverse, high-quality, and fully textured 3D worlds with fine-grained control over their layout and appearance. SynCity creates worlds by autoregressively generating tiles on a grid, which can be scaled to arbitrary sizes. By accounting for local context when generating individual tiles and applying a novel blending operation, the tiles smoothly integrate with one another, creating seamless and coherent scenes. SynCity is flexible: It can either generate worlds from a brief 'world' text prompt, but also allows fine-grained control of individual tiles via tile-specific instructions. Despite offering this high degree of control, SynCity maintains an overall stylistic and thematic consistency of the generated world. The rich detail of the generated worlds can be fully explored, without being restricted to a single '3D bubble' as in many prior works.

We have demonstrated the effectiveness of off-the-shelf generation by utilizing pre-trained language, 2D, and 3D generators through carefully designed prompting strategies. This eliminates the need to retrain any of these components, which would be complicated by the lack of large-scale scene datasets. Nevertheless, we expect that once these do become available, fine-tuning some components could result in further improvement of the method's performance, and would further simplify the alignment and rebasing steps of the pipeline. Future work could also consider relaxing the rigid grid structure and establishing a greater coherence between tiles. In terms of structure, the tiles could be randomly shifted and scaled. To ensure a coherent global theme that is carried into fine-grained local details, a coarse-to-fine approach could be employed. Here, a theme could inform a coarse representation of a grid, which then influences the generation of individual tiles on a local level.

Ethics. For details on ethics, data protection, and copyright, please see https://www.robots.ox.ac.uk/~vedaldi/research/union/ethics.html.

Acknowledgments. The authors of this work are supported by ERC 101001212-UNION, AIMS EP/S024050/1, and Meta Research.

References

[1] AlimamaCreative. Flux-controlnet-inpainting. https://github.com/alimama-creative/FLUX-Controlnet-Inpainting, 2024. GitHub repository. 7

[2] Miguel Ángel Bautista et al. GAUDI: A neural architect for immersive 3D scene generation. *arXiv.cs*, abs/2207.13751, 2022. 3

[3] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel and Arthur Gretton. Demystifying mmd gans. *arXiv preprint arXiv:1801.01401*, 2018. 7

[4] Ron Brinkmann. *The art and science of digital compositing: Techniques for visual effects, animation and motion graphics*. Morgan Kaufmann, 2008. 5

[5] Shengqu Cai, Eric Ryan Chan, Songyou Peng, Mohamad Shahbazi, Anton Obukhov, Luc Van Gool and Gordon Wetzstein. DiffDreamer: Towards consistent unsupervised single-view scene extrapolation with conditional diffusion models. In *Proc. ICCV*, 2023. 3

[6] Lucy Chai, Richard Tucker, Zhengqi Li, Phillip Isola and Noah Snavely. Persistent nature: A generative model of unbounded 3d worlds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20863–20874, 2023.

[7] Zhaoxi Chen, Guangcong Wang and Ziwei Liu. Scenedreamer: Unbounded 3d scene generation from 2d image collections. *IEEE transactions on pattern analysis and machine intelligence*, 45(12):15562–15576, 2023. 3

[8] Jaeyoung Chung, Suyoung Lee, Hyeongjin Nam, Jaerin Lee and Kyoung Mu Lee. LucidDreamer: Domain-free generation of 3d gaussian splatting scenes. In *arXiv*, 2023. 2

[9] Dana Cohen-Bar, Elad Richardson, Gal Metzer, Raja Giryes and Daniel Cohen-Or. Set-the-scene: Global-local training for generating controllable NeRF scenes. In *Proc. ICCV Workshops*, 2023. 3

[10] Deemos. Rodin text-to-3D gen-1 (0525) v0.5, 2024. 2

[11] Matt Deitke et al. Objaverse-XL: A universe of 10M+ 3D objects. *CoRR*, abs/2307.05663, 2023. 3

[12] Paul Engstler, Andrea Vedaldi, Iro Laina and Christian Rupprecht. Invisible stitch: Generating smooth 3D scenes with depth inpainting. In *Proceedings of the International Conference on 3D Vision (3DV)*, 2025. 2

[13] Rafail Fridman, Amit Abecasis, Yoni Kasten and Tali Dekel. Scenescape: Text-driven consistent scene generation. *CoRR*, abs/2302.01133, 2023. 2

[14] Ruiqi Gao, Aleksander Holynski, Philipp Henzler, Arthur Brussee, Ricardo Martin-Brualla, Pratul Srinivasan, Jonathan T. Barron and Ben Poole. CAT3D: create anything in 3d with multi-view diffusion models. *arXiv*, 2405.10314, 2024. 1

[15] Daniel Gatis. rembg, 2025. 5

[16] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Proc. NeurIPS*, 2017. 7

[17] Lukas Höllein, Ang Cao, Andrew Owens, Justin Johnson and Matthias Nießner. Text2Room: Extracting textured 3D meshes from 2D text-to-image models. In *Proc. ICCV*, 2023. 2

[18] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 conference papers*, pages 1–11, 2024. 2

[19] Zehuan Huang, Yuan-Chen Guo, Xingqiao An, Yunhan Yang, Yangguang Li, Zi-Xin Zou, Ding Liang, Xihui Liu, Yan-Pei Cao and Lu Sheng. Midi: Multi-instance diffusion for single image to 3d scene generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 23646–23657, 2025. 3

[20] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler and George Drettakis. 3D Gaussian Splatting for real-time radiance field rendering. *Proc. SIGGRAPH*, 42(4), 2023. 2

[21] Black Forest Labs. Flux. https://github.com/black-forest-labs/flux, 2023. 2

[22] World Labs. Generating worlds, 2024. 2

[23] Jiabao Lei, Jiapeng Tang and Kui Jia. RGBD2: generative scene synthesis via incremental view inpainting using RGBD diffusion models. In *Proc. CVPR*, 2023. 2

[24] Jiaxin Li, Zijian Feng, Qi She, Henghui Ding, Changhu Wang and Gim Hee Lee. MINE: towards continuous depth MPI with NeRF for novel view synthesis. In *Proc. ICCV*, 2021. 2

[25] Jiahao Li, Hao Tan, Kai Zhang, Zexiang Xu, Fujun Luan, Yinghao Xu, Yicong Hong, Kalyan Sunkavalli, Greg Shakhnarovich and Sai Bi. Instant3D: Fast text-to-3D with sparse-view generation and large reconstruction model. *Proc. ICLR*, 2024. 2

[26] Weiyu Li, Jiarui Liu, Rui Chen, Yixun Liang, Xuelin Chen, Ping Tan and Xiaoxiao Long. CraftsMan: high-fidelity mesh generation with 3d native generation and interactive geometry refiner. *arXiv*, 2405.14979, 2024. 2, 3

[27] Yangguang Li et al. TripoSG: high-fidelity 3D shape synthesis using large-scale rectified flow models. *arXiv*, 2502.06608, 2025. 2, 3

[28] Zhengqi Li, Qianqian Wang, Noah Snavely and Angjoo Kanazawa. Infinitenature-zero: Learning perpetual view generation of natural scenes from single images. In *European Conference on Computer Vision*, pages 515–534. Springer, 2022. 3

[29] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu and Tsung-Yi Lin. Magic3D: High-resolution text-to-3D content creation. *arXiv.cs*, abs/2211.10440, 2022. 3

[30] Chieh Hubert Lin, Hsin-Ying Lee, Willi Menapace, Menglei Chai, Aliaksandr Siarohin, Ming-Hsuan Yang and Sergey Tulyakov. Infinicity: Infinite-scale city synthesis. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 22808–22818, 2023. 3

[31] A Liu, R Tucker, V Jampani, A Makadia and N Snavely.... Infinite nature: Perpetual view generation of natural scenes from a single image. In *Proc. ICCV*, 2021. 2, 3

[32] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3D object. In *Proc. ICCV*, 2023. 3

[33] Luke Melas-Kyriazi, Iro Laina, Christian Rupprecht, Natalia Neverova, Andrea Vedaldi, Oran Gafni and Filippos Kokkinos. IM-3D: Iterative multiview diffusion and reconstruction for high-quality 3D generation. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024. 1

[34] Quan Meng, Lei Li, Matthias Nießner and Angela Dai. LT3SD: latent trees for 3D scene diffusion. *arXiv*, 2409.08215, 2024. 2, 3

[35] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng and Abhishek Kar. Local light field fusion: practical view synthesis with prescriptive sampling guidelines. *Proc. SIGGRAPH*, 38(4), 2019. 2

[36] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proc. ECCV*, 2020. 2

[37] OpenAI et al. GPT-4 technical report. *arXiv*, 2303.08774, 2024. 2, 4

[38] Hao Ouyang, Kathryn Heal, Stephen Lombardi and Tiancheng Sun. Text2Immersion: Generative immersive scene with 3D gaussians. *arXiv.cs*, abs/2312.09242, 2023. 2

[39] Jack Parker-Holder et al. Genie 2: A large-scale foundation world model, 2024. 8

[40] Ben Poole, Ajay Jain, Jonathan T. Barron and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. In *Proc. ICLR*, 2023. 1

[41] Guocheng Qian et al. Magic123: One image to high-quality 3D object generation using both 2D and 3D diffusion priors. *arXiv.cs*, abs/2306.17843, 2023. 3

[42] Amit Raj et al. DreamBooth3D: subject-driven text-to-3D generation. In *Proc. ICCV*, 2023. 3

[43] Chris Rockwell, David F. Fouhey and Justin Johnson. PixelSynth: Generating a 3D-consistent experience from a single image. In *Proc. ICCV*, 2021. 2

[44] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. CVPR*, 2022. 1

[45] Kyle Sargent et al. ZeroNVS: Zero-shot 360-degree view synthesis from a single real image. *arXiv.cs*, abs/2310.17994, 2023. 2, 3

[46] Junyoung Seo, Kazumi Fukuda, Takashi Shibuya, Takuya Narihira, Naoki Murata, Shoukang Hu, Chieh-Hsin Lai, Seungryong Kim and Yuki Mitsufuji. GenWarp: single image to novel views with semantic-preserving generative warping. *arXiv*, 2405.17251, 2024. 2

[47] Yu Shang, Yuming Lin, Yu Zheng, Hangyu Fan, Jingtao Ding, Jie Feng, Jiansheng Chen, Li Tian and Yong Li. Urbanworld: An urban world model for 3d city generation. *arXiv preprint arXiv:2407.11965*, 2024. 3

[48] Yichun Shi, Peng Wang, Jianglong Ye, Mai Long, Kejie Li and Xiao Yang. MVDream: Multi-view diffusion for 3D generation. In *Proc. ICLR*, 2024. 1, 3

[49] Meng-Li Shih, Shih-Yang Su, Johannes Kopf and Jia-Bin Huang. 3d photography using context-aware layered depth inpainting. In *Proc. CVPR*, 2020. 2

[50] Jaidev Shriram, Alex Trevithick, Lingjie Liu and Ravi Ramamoorthi. RealmDreamer: text-driven 3d scene generation with inpainting and depth diffusion. In *Proc. 3DV*, 2025. 2

[51] Pratul P Srinivasan, Richard Tucker, Jonathan T Barron, Ravi Ramamoorthi, Ren Ng and Noah Snavely. Pushing the boundaries of view extrapolation with multiplane images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 175–184, 2019. 2

[52] Gabriela Ben Melech Stan et al. LDM3D: Latent diffusion model for 3D. *arXiv.cs*, 2305.10853, 2023. 3

[53] Stanislaw Szymanowicz, Christian Rupprecht and Andrea Vedaldi. Viewset diffusion: (0-)image-conditioned 3D generative models from 2D data. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2023. 1

[54] Jiaxiang Tang, Zhaoxi Chen, Xiaokang Chen, Tengfei Wang, Gang Zeng and Ziwei Liu. LGM: Large multi-view Gaussian model for high-resolution 3D content creation. *arXiv*, 2402.05054, 2024. 3

[55] Richard Tucker and Noah Snavely. Single-view view synthesis with multiplane images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 551–560, 2020. 2

[56] Shubham Tulsiani, Richard Tucker and Noah Snavely. Layer-structured 3d scene inference via view synthesis. In *Proc. ECCV*, 2018. 2

[57] Guangcong Wang, Peng Wang, Zhaoxi Chen, Wenping Wang, Chen Change Loy and Ziwei Liu. PERF: panoramic neural radiance field from a single panorama. *tpami*, 2024. 3

[58] Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su and Jun Zhu. Prolificdreamer: High-fidelity and diverse text-to-3d generation with variational score distillation. *Advances in Neural Information Processing Systems*, 36:8406–8441, 2023. 2

[59] Olivia Wiles, Georgia Gkioxari, Richard Szeliski and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *Proc. CVPR*, 2020. 2

[60] Zhennan Wu et al. BlockFusion: Expandable 3D scene generation using latent tri-plane extrapolation. *arXiv.cs*, 2024. 2, 3, 7, 14

[61] Jianfeng Xiang, Jiaolong Yang, Binbin Huang and Xin Tong. 3D-aware image generation using 2D diffusion models. *arXiv.cs*, abs/2303.17905, 2023. 2

[62] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. *arXiv preprint arXiv:2412.01506*, 2024. 6

[63] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong and Jiaolong Yang. Structured 3D latents for scalable and versatile 3d generation. *arXiv*, 2412.01506, 2024. 2, 3, 5

[64] Haozhe Xie, Zhaoxi Chen, Fangzhou Hong and Ziwei Liu. Citydreamer: Compositional generative model of unbounded 3d cities. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9666–9675, 2024. 3

[65] Yinghao Xu, Zifan Shi, Wang Yifan, Hansheng Chen, Ceyuan Yang, Sida Peng, Yujun Shen and Gordon Wetzstein.

GRM: Large gaussian reconstruction model for efficient 3D reconstruction and generation. *arXiv*, 2403.14621, 2024. 2

[66] Kaixin Yao, Longwen Zhang, Xinhao Yan, Yan Zeng, Qixuan Zhang, Lan Xu, Wei Yang, Jiayuan Gu and Jingyi Yu. Cast: Component-aligned 3d scene reconstruction from an rgb image. *ACM Transactions on Graphics (TOG)*, 44(4): 1–19, 2025. 3

[67] Hong-Xing Yu et al. Wonderjourney: Going from anywhere to everywhere. *arXiv.cs*, abs/2312.03884, 2023. 2

[68] Hong-Xing Yu, Haoyi Duan, Charles Herrmann, William T Freeman and Jiajun Wu. Wonderworld: Interactive 3d scene generation from a single image. *arXiv preprint arXiv:2406.09394*, 2024. 3

[69] Biao Zhang, Jiapeng Tang, Matthias Niessner and Peter Wonka. 3DShape2VecSet: A 3D shape representation for neural fields and generative diffusion models. In *ACM Transactions on Graphics*, 2023. 2

[70] Jingbo Zhang, Xiaoyu Li, Ziyu Wan, Can Wang and Jing Liao. Text2NeRF: Text-driven 3D scene generation with neural radiance fields. *arXiv.cs*, abs/2305.11588, 2023. 2

[71] Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu and Jingyi Yu. Clay: A controllable large-scale generative model for creating high-quality 3d assets. *ACM Transactions on Graphics (TOG)*, 43(4):1–20, 2024. 3

[72] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR*, pages 586–595, 2018. 7, 13

[73] Zibo Zhao et al. Hunyuan3d 2.0: Scaling diffusion models for high resolution textured 3d assets generation. *arXiv preprint arXiv:2501.12202*, 2025. 3

SynCity: Training-Free Generation of 3D Worlds

Supplementary Material

A. Appendix

A.1. Language Model Prompting Details

While the prompt p can be constructed manually, an LLM may also be employed. To help it understand the task, we utilize the following prompt for ChatGPT o3-mini-high:

> Assume you had access to an AI model that can generate small-scale cities on an isometric grid by creating individual tiles. For each of these tiles (identified by their 2D position), a short but expressive text prompt has to be provided. Additionally, a global prompt is used, which provides context, lighting, time of day, as well as the art style. The prompts of the tiles can be generic but they might have a semantic connection to neighbouring tiles (such that a river can flow through the city on multiple tiles). The format for the instructions to the AI model is JSON. Consider the following example: `<TEMPLATE>` The art style and perspective mentioned in the prompt should be maintained. The rest may be freely adapted. There are no limits to the setting, the sky is your limit. Now, please generate a 3×3 grid.

In this prompt, a template or a 'seed' prompt is provided. We use a simple JSON file, as exemplified in Fig. 13.

```
{"tiles": [
{ "prompt": "ancient stone bridge over a stream
    ", "x": 0, "y": 0 },
{ "prompt": "lively stream past mossy banks", "
    x": 1, "y": 0 },
{ "prompt": "serene pond reflecting moonlight",
     "x": 0, "y": 1 },
{ "prompt": "bustling medieval market street",
    "x": 1, "y": 1 } ],
"prompt": "{tile_prompt}, medieval setting,
    isometric view, glowing lanterns, soft
    shading, vibrant colors, detailed textures"
    }
```

Figure 13. Example JSON file to describe each tile in a 2×2 world.

A.2. 2D Prompting Details

In Sec. 3.2, we describe how tiles are generated in the context of those that already exist. There is a special case that we address separately, where context has to be bootstrapped, namely tiles $\mathcal{L} := \{(x, y) \in \mathcal{T} : x = 0 \wedge y > 0\}$. Due to our build order and the trimming of obstructing 3D geometry, these tiles might lack sufficient contextual cues. As a remedy, we temporarily provide context with a previously generated tile: For a tile $(0, y) \in \mathcal{L}$, we duplicate the tile $(0, y-1) \in \mathcal{T}$ and place the copy at position $(-1, y)$. During inpainting, this tile serves to provide context in terms of scale and general appearance. Once inpainting is completed, this copy is removed.

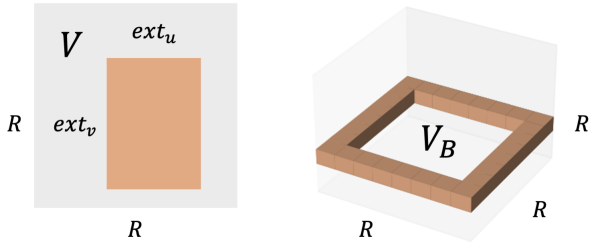

Figure 14. **Tile geometry validation.** To evaluate the geometric properties of a reconstructed tile, we examine the occupancy grid $V \in \{0,1\}^{R \times R \times R}$ generated by TRELLIS. Activated voxels are shown in orange (■). *Left:* The extent of an object at height w (slice visualized in 2D). *Right:* An example of a 3D tile base template V_B.

A.3. 3D Geometry Validation Details

TRELLIS is a two-stage method that produces an occupancy volume $V \in \{0,1\}^{R \times R \times R}$ in the first stage (before the 3D Gaussian mixture is output), where $R = 64$ is the resolution of the grid. To perform geometric validation, we utilize this occupancy volume, which captures the rough 3D geometry of the tile, and check that it conforms to the desired geometry.

First, we test whether the reconstructed tile is supported by a square by computing its 2D rectangular footprint and ensuring that the latter is sufficiently large and isotropic.

To this end, let (u, v, w) index the $R \times R \times R$ voxel grid, where u and v correspond to world directions x and y. Let $(u_{\min}, u_{\max}, v_{\min}, v_{\max})$ be the bounding box containing all the active voxels at height w.* Let $\text{ext}_u = \max\{0, 1 + u_{\max} - u_{\min}\}$ be the width of the bounding box and ext_v its height. We discard the tile if the area is too small, *i.e.*, if $\text{ext}_u \cdot \text{ext}_v < (R/2)^2$. We also discard it if it is not square, *i.e.*, if $\min\{\text{ext}_u, \text{ext}_v\} / \max\{\text{ext}_u, \text{ext}_v\} < \alpha = 1$.

Second, we check that the base added in 2D during the 'rebasing' step has been faithfully reconstructed in 3D.

*For instance, $u_{\min} = \min\{u : \exists v, w : V(u, v, w) = 1\}$.

We define a 3D tile base template, V_B. Let $u_{\min}(w)$ be the minimum u of the bounding box that contains the volume slice at height w, and define $u_{\max}(w)$ and so on in a similar manner, so for instance $u_{\min}(w) = \min\{u : \exists v : V(u, v, w) = 1\}$. Let w^* be the height at which the base is the largest, *i.e.*, $w^* = \operatorname{argmax}_w \operatorname{ext}_u(w) \cdot \operatorname{ext}_v(w)$. Then, V_B is the indicator function of the voxels (u, v, w) such that $w = w^*$ and

$$\max\left\{\frac{|2u - u_{\max}(w^*) - u_{\min}(w^*)|}{u_{\max}(w^*) - u_{\min}(w^*)}, \frac{|2v - v_{\max}(w^*) - v_{\min}(w^*)|}{v_{\max}(w^*) - v_{\min}(w^*)}\right\} = 1.$$

Note that the template V_B is constructed adaptively to match the input tile V.

We discard a generated tile if $(V \cdot V_B)/(V_B \cdot V_B) < \beta = 0.95$, where $\cdot$ denotes the inner product of tensors.

A.4. 3DGS Post-Processing Details

3D cropping, resizing, and centering. Given the 3D Gaussian mixture $G(x, y)$ initially output by the 3D generator, we first identify the extent of the tile 'proper' (discounting the extended base). We consider the xy footprint of the tile (*i.e.*, we look at the tile from above) and seek to identify four cuts (from the left, right, top, and bottom) that define an axis-aligned rectangle strictly containing the tile. For example, to determine the location of the left cut x^*, we consider slices $V_x = \{(x', y', z') \in \mathbb{R}^3 : x - \delta \leq x' < x + \delta\}$. We find the 3D Gaussians whose centers fall within V_x and compute their average color c_x. Then, we compute the distance $d(x) = \|c_x - c_{x_{\min}}\|$, where $c_{x_{\min}}$ is the average color of the leftmost slice (used as a reference). We set $x^* = \min\{x : d(x) > \tau\}$, where τ is a threshold, which corresponds to the slice that transitions from the 'background' color to something else.

We find the four cuts in this manner, keep only the Gaussians contained in the resulting rectangular footprint, and recenter and resize this footprint to fill the standard tile size.

Additionally, the base allows us to determine the position of the tile's surface: As TRELLIS centers objects vertically, the ground surface level of any two tiles may vary. We use the average height of the tile's four corners to determine the position of the surface, allowing us to align it with others.

3D reorientation. TRELLIS generates the 3D object with an arbitrary orientation with respect to the input image $\tilde{I}(x, y)$. However, the tile must be inserted with the correct orientation in the 3D world; otherwise, the continuity between tiles, which the inpainting method of Fig. 4 encourages, will be lost. In practice, the ambiguity is limited to 90-degree rotations around the vertical axis and is straightforward to resolve.* To do so, we test four possible 90-degree rotations of the tile around the vertical axis, render the corresponding views, and compare them to $\tilde{I}(x, y)$ using the LPIPS [72] loss. The rotation that minimizes the loss is taken as the correct orientation.

*This is likely due to the implicit bias in the TRELLIS training set, which consists of synthetic 3D objects that are almost invariably axis-aligned.

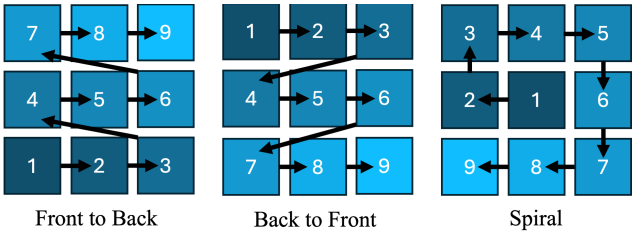

Figure 15. **Build order options.** Illustrations of build orders that appear practical when generating a 3×3 grid. Our method employs the "Front to Back" approach.

Figure 16. **Masking schemes for different build orders.** Depending on the build order (see Fig. 15), the masking scheme (Section 3.2) needs to be adapted. Top row: front to back, center: back to front, bottom: spiral.

A.5. Different Build Order Schemes

We experimented with different build orders (Fig. 15) and masking schemes (Fig. 16). There is a trade-off between how much context the inpainter is given, how much is inpainted, and how easy it is to extract the new tile. Build orders other than the front-to-back scheme either limit the ease of tile extraction (*e.g.*, back to front, center row) or require more occlusion trimming, which reduces context (*e.g.*, spiral, bottom).

A.6. Ablation Details

Building a Grid. In the following, we present results from our experiments attempting to generate a large scene non-iteratively. Here, we generate a single image with Flux,

which is used as conditioning for TRELLIS to directly create the desired scene.

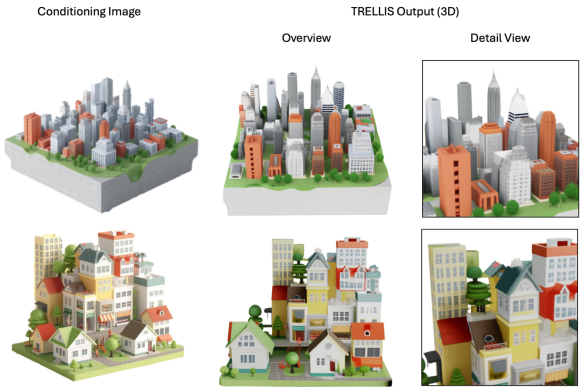

Figure 17. **Non-iterative city building.** Conditioning images generated by Flux (left) are directly used to construct a large-scale scene with TRELLIS (center). While the generated 3D structures are visually appealing, their level of detail (right) is limited. The first row uses a generic prompt for the conditioning image (i.e., "a city scene on top of a base"), whereas the second row employs a more detailed prompt specifying the layout (i.e., "a house in the bottom left corner", "a pharmacy in the top right corner").

In the first set of experiments, we do not use our 2D prompting design. To obtain an isolated 3D object that can be generated by TRELLIS, we use prompts with the prefix "a 3D object of". We show these results in Fig. 17. While the generated objects are visually appealing, they have several limitations: (i) The resolution of the conditioning image and the 3D structures TRELLIS can generate is limited, making this approach unsuitable for arbitrarily large scenes. (ii) Due to the lack of precise control over the base structure, the result cannot be easily extended or edited. (iii) The layout instructions are mostly ignored, severely limiting the level of control over the generation.

For the second set of experiments, we use our 2D prompting design along with the Flux ControlNet for inpainting (Fig. 18). However, with this setup, the quality of the results is not improved. The layout instructions in the prompt are again mostly ignored.

Querying Flux to generate large-scale scenes directly has not been successful in our experiments, prompting the need for our grid-based method, which allows fine-grained layout and appearance control for each tile.

A.7. Additional Qualitative Results

In Figs. 21 to 24, we show additional results of our method. By leveraging a pre-trained 2D image generator trained on a very large dataset, we are able to generate highly diverse scenes. Thanks to our fine-grained control at the tile level, we can generate interesting patterns, such as a transition between seasons across a grid (observe the largest grid in Fig. 24).

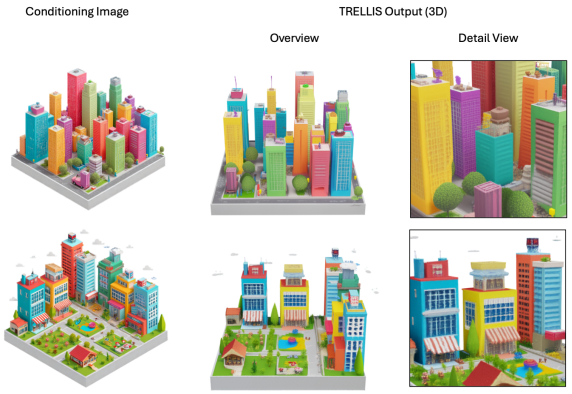

Figure 18. **Non-iterative city building using 2D prompting.** Conditioning images generated by Flux (left) are directly used to construct a large-scale scene with TRELLIS (center). While visually appealing, the resulting structures lack detail. The first row uses a generic prompt for the conditioning image (i.e., "a vibrant city scene"), whereas the second row employs a more detailed prompt specifying the layout (i.e., "a house in the bottom left corner", "a pharmacy in the top right corner").

SynCity Scene Continuity Win Rate (%) vs:				
LucidDreamer	Invisible Stitch	RealmDreamer†	WonderWorld	WonderWorld†
57.1	95.2	100.0	85.7	47.6

Table 4. **User preference results.** We forced a binary choice between image sequence pairs showing steps into a generated scene.

Geometry. In Fig. 19, we show the geometry underlying a scene generated by our method. Compared to previous methods, such as [60], our method generates high-fidelity geometry without any holes.

A.8. Additional Comparisons

User preference. To validate our claim that the fully realized 3D worlds generated by our method are more convincing when stepping into them, we conducted a user study comparing them to the 'bubble' worlds created by other methods. We presented users ($n = 22$) with pairs of image sequences depicting a few small steps *inside* the bubble († methods generated the scene from the GT camera poses used to render the images in the sequence, which is a significantly easier task). Note that the total step distance is equivalent to the extent of only a *single* tile in our world. Users were asked to pick which sequence appeared visually more consistent. Prior methods start with a more "realistic" version of a SynCity rendering (to stay within their training distribution). See Fig. 20 for an example image sequence pair.

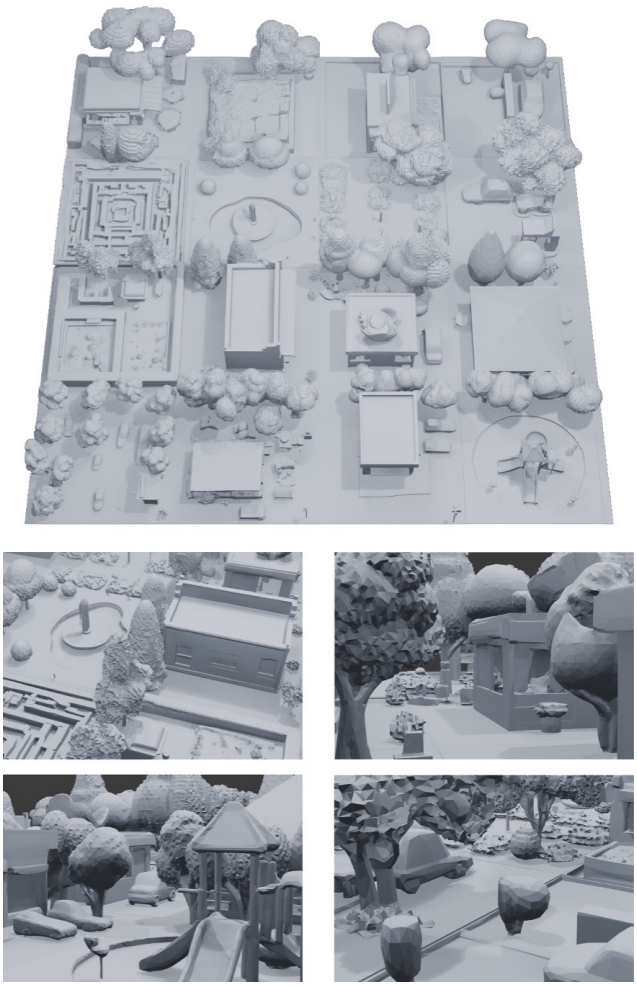

Figure 19. Views of the suburban scene in Fig. 1 as an untextured mesh.

Figure 20. **User study sample.** An image sequence pair presented to users in the user preference study (Tab. 4). Top: WonderWorld†, bottom: our method.

A.9. Limitations

While our method allows the creation of large and diverse scenes, there are some limitations to be addressed in future work.

Atomic tiles. Although we inpaint tiles conditioned on their surroundings, they remain individual units. While structures spanning multiple tiles can be created, this requires harmonious cooperation between Flux and TRELLIS.

Use of heuristics. To determine the ground surface height for each tile and to remove the base added during rebasing, we employ heuristics. While these are carefully designed with fallback mechanisms, they are not infallible.

Inherited limitations. As our method builds on top of Flux and TRELLIS, their limitations also apply to ours. During our experiments, we observed that, despite good inpainting results, TRELLIS sometimes only vaguely adheres to the conditioning image in terms of appearance, particularly color. As a result, transitions between tiles might not look perfectly smooth, even if they were generated that way in the inpainting result.

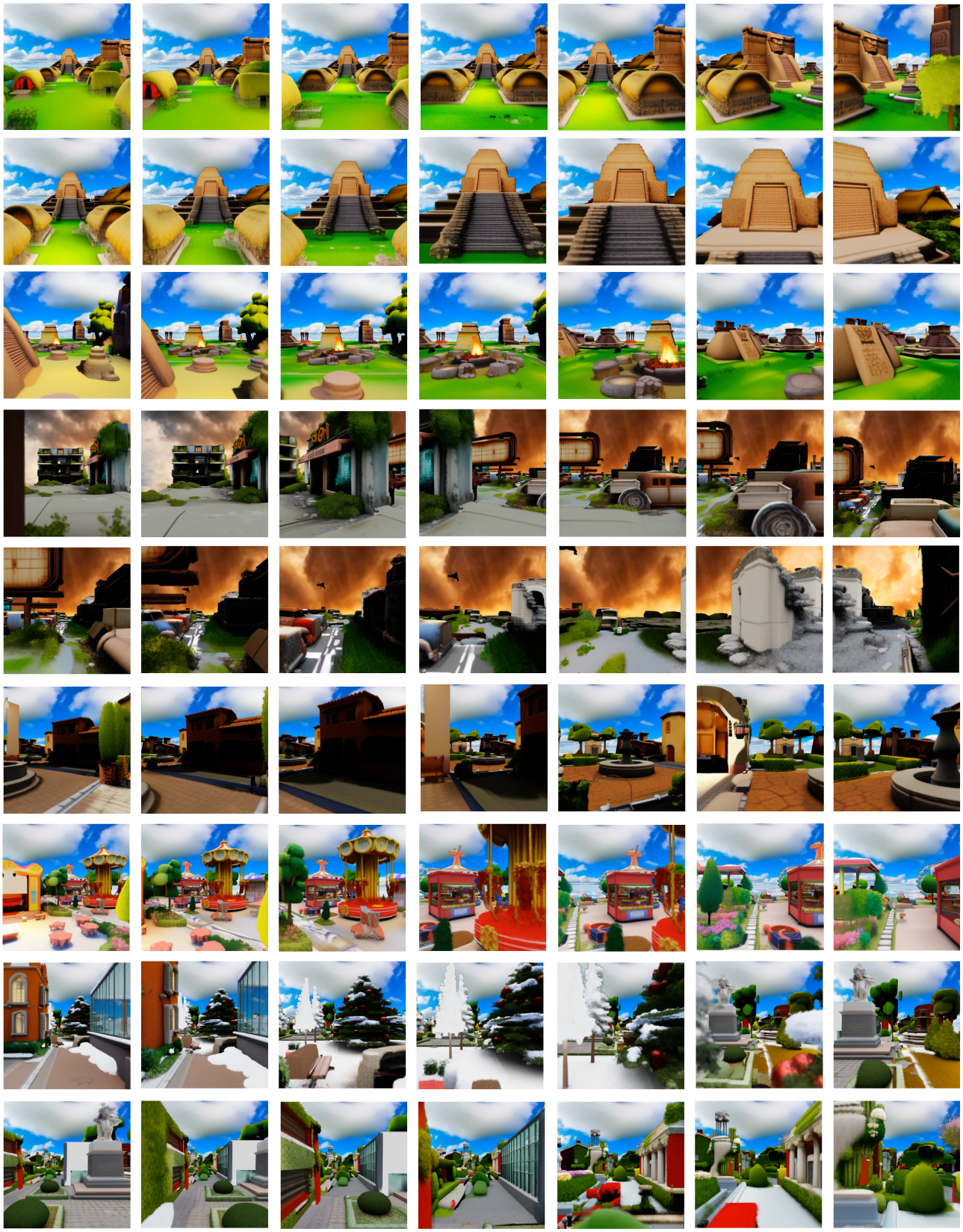

Figure 21. **Exploring a 3D world.** Trajectories illustrating the exploration of the generated 3D worlds are shown. A skybox has been added to enhance visual appeal.

Figure 22. **Generated scenes.** We show scenes generated with the same prompts, but different seeds in 2D inpainting.

Figure 23. **Generated scenes.**

Figure 24. **Generated scenes.** Our method can easily generate large scenes. Further, interesting patterns can be injected thanks to fine-grained control over each tile. *Top:* The scene transitions in season, from winter to spring to summer to autumn. *Bottom:* The scenery transitions from a city-like to a rural environment.