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.

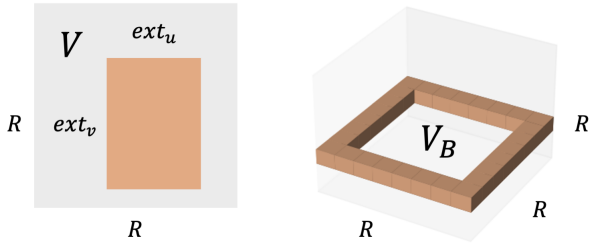

Figure 14. **Tile geometry validation.** To evaluate the geometric properties of a reconstructed tile, we examine the occupancy grid $V \in \{0,1\}^{R \times R \times R}$ generated by TRELLIS. Activated voxels are shown in orange (■). *Left:* The extent of an object at height w (slice visualized in 2D). *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.*, $w^* = \text{argmax}_w \, \text{ext}_u(w) \cdot \text{ext}_v(w)$. 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.

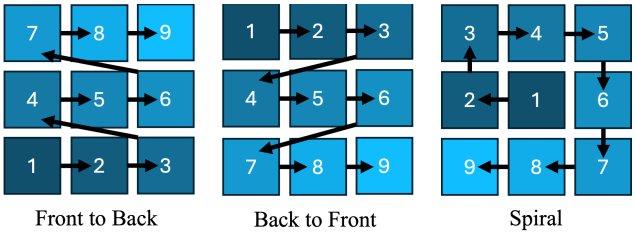

Figure 15. **Build order options.** Illustrations of build orders that appear practical when generating a 3×3 grid. Our method employs the "Front to Back" approach.

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.

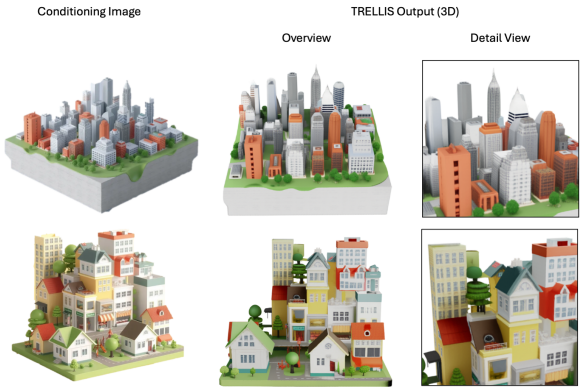

Figure 17. **Non-iterative city building.** Conditioning images generated by Flux (left) are directly used to construct a large-scale scene with TRELLIS (center). While the generated 3D structures are visually appealing, their level of detail (right) is limited. The first row uses a generic prompt for 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).

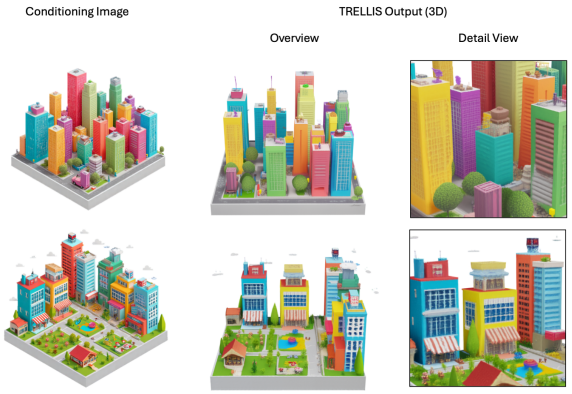

Figure 18. **Non-iterative city building using 2D prompting.** Conditioning images generated by Flux (left) are directly used to construct a large-scale scene with TRELLIS (center). While visually appealing, the resulting structures lack detail. The first row uses a generic prompt for the conditioning 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.

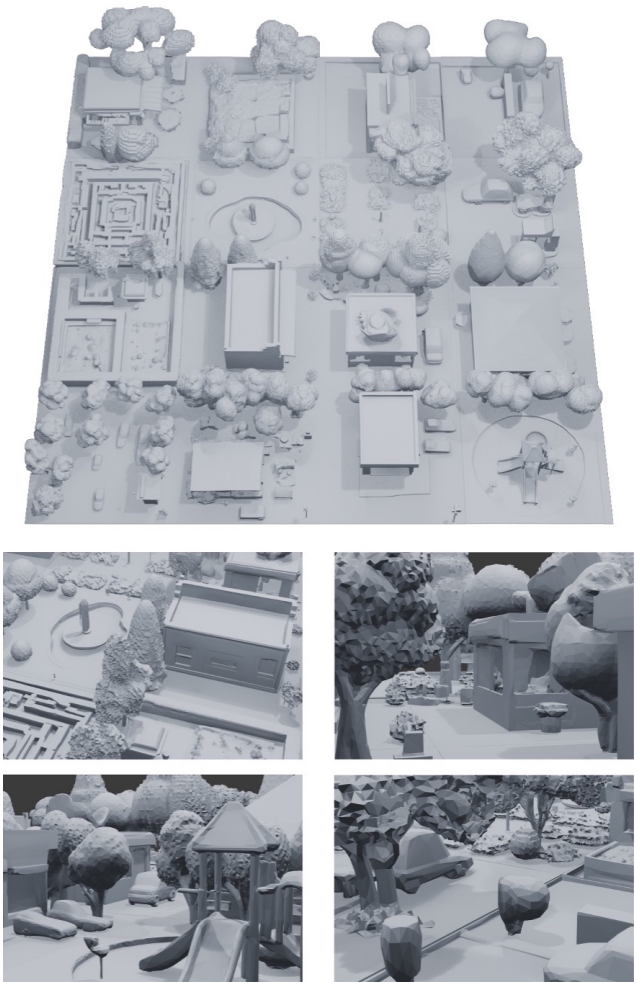

Figure 19. Views of the suburban scene in Fig. 1 as an untextured mesh.

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.

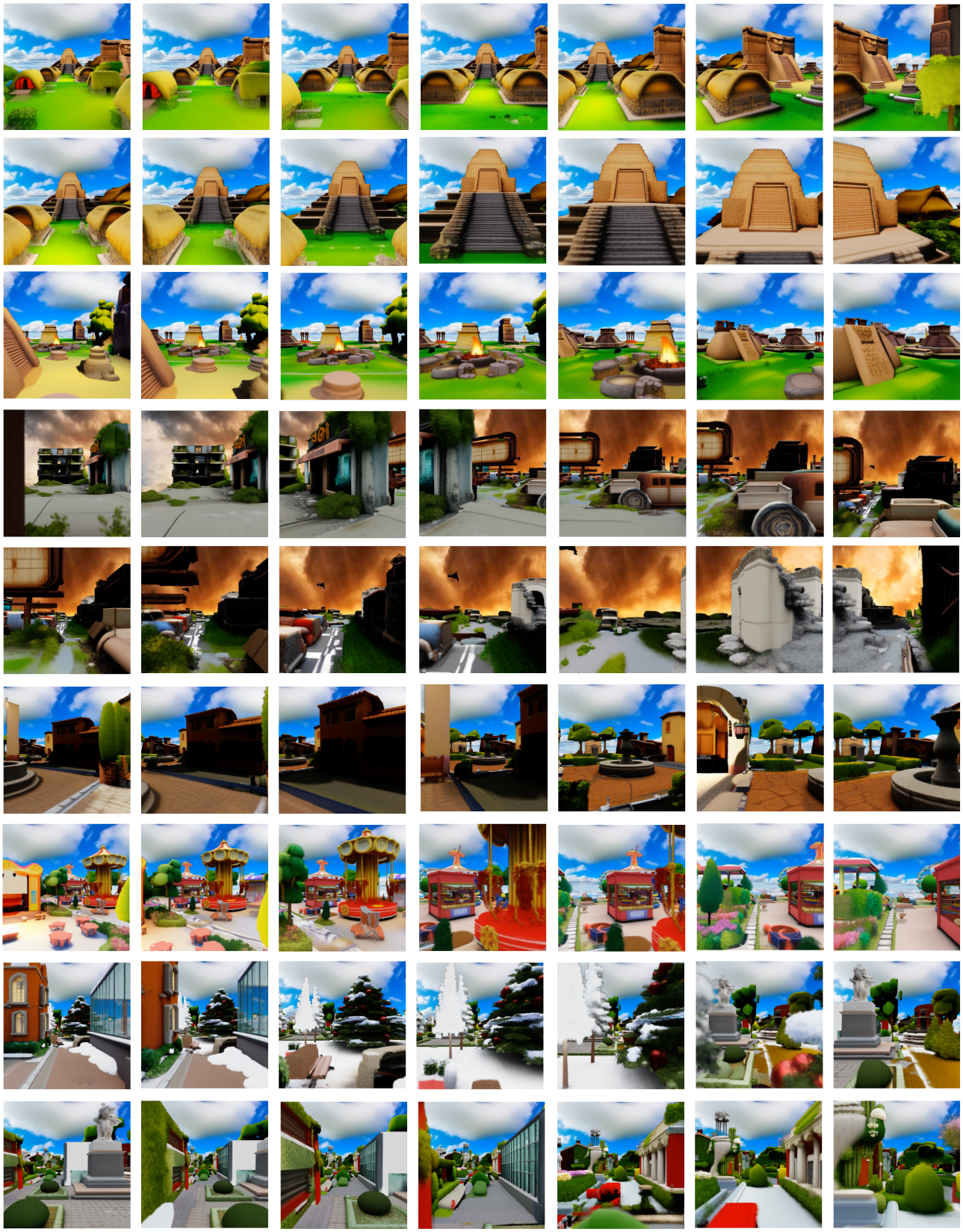

Figure 21. **Exploring a 3D world.** Trajectories illustrating the exploration of the generated 3D worlds are shown. A skybox has been added to enhance visual appeal.

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.