Supplemental Material
ExCellGen: Fast, Controllable, Photorealistic 3D Scene Generation from a Single Real-World Exemplar

Clément Jambon*
Massachusetts Institute of Technology
Cambridge, USA
cjambon@mit.edu

Changwoon Choi
Seoul National University
Seoul, South Korea
changwoon.choi00@gmail.com

Dongsu Zhang
Seoul National University
Seoul, South Korea
96lives@gmail.com

Olga Sorkine-Hornung
ETH Zurich
Zurich, Switzerland
sorkine@inf.ethz.ch

Young Min Kim
Seoul National University
Seoul, South Korea
youngmin.kim@snu.ac.kr

S.1. Generative Cellular Automata

The following section provides a full description of GCA (Section S.1.1), followed by details about its training strategy (Section S.1.2). Finally, we discuss the limitations of GCA's training strategy, which we encountered when adapting it to our method, and propose ways to mitigate them (Section S.1.3).

S.1.1. Full description of GCA

Generative Cellular Automata (GCA) [18, 19] represent shapes as a sparse grid of voxels $\mathbf{V} = \{\mathbf{p}_i\}_{i=1}^N$ with $\mathbf{p}_i \in \mathbb{Z}^3$ the positional index of an occupied voxel. Additionally, continuous GCA [19] augment features $\mathbf{Z} = \{\mathbf{z}_i\}_{i=1}^N$ for each voxel, effectively modeling the joint distribution $X = (\mathbf{V}, \mathbf{Z})$. GCA generates shapes according to a learnable transition kernel:

$$X^{t+1} \sim p_\theta(\cdot \mid X^t)$$

which is factorized into a term predicting first the position of new voxels, and then their corresponding features:

$$p_\theta(X^{t+1} \mid X^t) = p_\theta(\mathbf{V}^{t+1} \mid X^t)\, p_\theta(\mathbf{Z}^{t+1} \mid X^t, \mathbf{V}^{t+1})$$

At a given time step t, only transitions in the direct neighborhood of occupied voxels are considered. The neighborhood of a set of voxels $\mathbf{V}$ is defined as $\mathcal{N}(\mathbf{V}) = \{\mathbf{p}' \in \mathbb{Z}^3 \mid d(\mathbf{p}, \mathbf{p}') \leq k,\ \mathbf{p} \in \mathbf{V}\}$ where d is the L1 distance and k defines the size of the chosen neighborhood.

To keep inference tractable, GCA builds on the intuition behind cellular automata by factorizing transition kernels independently for each voxel and only allowing transitions in the *local* neighborhood of existing states. Under these assumptions, Equation (S.1.1) can be rewritten:

$$p(X^{t+1} \mid X^t) = \prod_{\mathbf{p} \in \mathcal{N}(\mathbf{V}^t)} p_\theta(o_\mathbf{p} \mid X^t)\, p_\theta(\mathbf{z}_\mathbf{p} \mid X^t, o_\mathbf{p}) \tag{1}$$

where the occupancy $o_\mathbf{p} \in \{0, 1\}$ specifies whether a voxel with coordinate $\mathbf{p} \in \mathbb{Z}^3$ will be part of the next voxel set at time $t+1$ i.e., $\mathbf{p} \in \mathbf{V}^{t+1}$. Binary occupancy transition kernels are parameterized as Bernoulli distributions while feature kernels are parameterized as Gaussian distributions if the voxel is occupied:

$$p_\theta(o_\mathbf{p} | X^t) = \text{Ber}(\lambda_{\theta,\mathbf{p}}),$$

$$p_\theta(\mathbf{z}_\mathbf{p} | X^t, o_\mathbf{p}) = \begin{cases} \delta_0 & \text{if } o_\mathbf{p} = 0 \\ N(\boldsymbol{\mu}_{\theta,\mathbf{p}},\, \sigma^t \mathbf{I}) & \text{if } o_\mathbf{p} = 1. \end{cases}$$

*Work done at Seoul National University and ETH Zurich.

where $\lambda_{\theta,\mathbf{p}} \in [0,1]$ and $\boldsymbol{\mu}_{\theta,\mathbf{p}}$ are directly predicted by the U-Net and σ^t is progressively annealed during sampling. In practice, Equation (1) is implemented with generalized convolutions [4].

S.1.2. Infusion Training and Loss

GCA is trained through a process called *Infusion* [2]. Intuitively, starting from a state X^0, *Infusion* supervises the learned transition kernel $p_\theta(X^{t+1} \mid X^t)$ to stay close to an "infused kernel" q_θ^t biased towards a target shape X which is factorized similarly to the transition kernel in Equation (1):

$$q_\theta^t(X^{t+1} \mid X^t, X) = \prod_{\mathbf{p} \in \mathcal{N}(\mathbf{V}^t)} q_\theta^t(o_\mathbf{p} \mid X^t, X)\; q_\theta^t(\mathbf{z}_\mathbf{p} \mid X^t, o_\mathbf{p}, X)$$

Intuitively, the conditional distributions of $\mathbf{p}$ and $\mathbf{z}$ are gradually biased towards the ground truth final shape X according to an infusion rate α^t which increases linearly with time step t, i.e., $\alpha^t = \min(\alpha_1 t + \alpha_0, 1)$, where $\alpha_1 > 0$:

$$q_\theta^t(o_\mathbf{p} \mid X^t, X) = Ber((1-\alpha^t)\lambda_{\theta,\mathbf{p}} + \alpha^t \mathbf{1}[\mathbf{p} \in X]), \tag{2}$$

$$q_\theta^t(\mathbf{z}_\mathbf{p} \mid X^t, \mathbf{p}, X) = \begin{cases} \delta_0 & \text{if } o_\mathbf{p} = 0 \\ N((1-\alpha^t)\boldsymbol{\mu}_{\theta,\mathbf{p}} + \alpha^t \mathbf{z}_\mathbf{p}^X,\ \sigma^t \mathbf{I}) & \text{if } o_\mathbf{p} = 1. \end{cases} \tag{3}$$

where $\mathbf{1}$ designates the indicator function and $\mathbf{I}$ the identity matrix.

To learn to generate $X \sim X^T$ from an initial state X^0, intermediate states $X^{1:T}$ are recursively sampled from the infused kernel $q_\theta(X^{t+1} \mid X^t, X)$. For each time step t, GCA is trained by minimizing the loss $\mathcal{L}_t$ defined as the KL divergence between the infused kernel and the transition kernel [19], which following the factorization defined in Equation (1) can be decomposed into:

$$\begin{aligned} \mathcal{L}_t &= D_{KL}(q_\theta(o_\mathbf{p}, \mathbf{z}_\mathbf{p}) \,||\, p_\theta(o_\mathbf{p}, \mathbf{z}_\mathbf{p})) \\ &= \sum_{\mathbf{p} \in \mathcal{N}(X^t)} \underbrace{D_{KL}(q_\theta(o_\mathbf{p} \mid X^t, X) \,||\, p_\theta(o_\mathbf{p} \mid X^t))}_{\mathcal{L}_o} + \\ &\quad q_\theta(o_\mathbf{p} = 1 \mid X^t, X)\, \underbrace{D_{KL}(q_\theta(\mathbf{z}_\mathbf{p} \mid X^t, X, o_\mathbf{p} = 1) \,||\, p_\theta(\mathbf{z}_\mathbf{p} \mid X^t, o_\mathbf{p} = 1))}_{\mathcal{L}_z}. \end{aligned} \tag{4}$$

Since $\mathcal{L}_o$ and $\mathcal{L}_z$ are the KL divergence between Bernoulli and normal distributions, respectively, $\mathcal{L}_t$ can be derived in closed form. In practice, the scale of $\mathcal{L}_z$ is much larger than that of $\mathcal{L}_o$ and $\mathcal{L}_z$ is downweighted by a factor λ_z. We follow Zhang *et al*. [19] and use $\lambda_z = 0.01$.

S.1.3. Discussion on GCA Training

While experimenting with GCA for single-exemplar training, we realized two main caveats of the loss defined by Equation (4).

Invalid variational lower bound. Zhang *et al*. [19] derived $\mathcal{L}_t$ through the evidence lower bound (ELBO). In what follows, we show that this holds, but the corresponding proposal distribution is intractable and not the one used during training. Let X be the target exemplar shape, we can bound the log-likelihood as follows:

$$\log p_\theta(X) \geq \mathbb{E}_{q_\theta(X^{0:T-1}|X)} \left[\log \frac{p_\theta(X^{0:T-1}, X)}{q_\theta(X^{0:T-1}|X)} \right] \tag{5}$$

$$\begin{aligned} &= \mathbb{E}_{q_\theta(X^{0:T-1}|X)} \left[\log \frac{p(X^0)}{q(X^0)} \right] + \mathbb{E}_{q_\theta(X^{0:T-1}|X)} \left[\sum_{t=0}^{T-2} \log \frac{p_\theta(X^{t+1}|X^t)}{q_\theta(X^{t+1}|X^t, X)} \right] \\ &\qquad + \mathbb{E}_{q_\theta(X^{0:T-1}|X)} \left[\log p(X|X^{T-1}) \right] \end{aligned} \tag{6}$$

$$\begin{aligned} &= \underbrace{\mathbb{E}_{q_\theta(X^0|X)} \left[\log \frac{p(X^0)}{q(X^0)} \right]}_{\text{(a)}} + \sum_{t=0}^{T-2} \underbrace{\mathbb{E}_{q_\theta(X^t, X^{t+1}|X)} \left[\log \frac{p_\theta(X^{t+1}|X^t)}{q_\theta(X^{t+1}|X^t, X)} \right]}_{\text{(b)}} \\ &\qquad + \underbrace{\mathbb{E}_{q_\theta(X^{T-1}|X)} \left[\log p(X|X^{T-1}) \right]}_{\text{(c)}} \end{aligned} \tag{7}$$

where Equation (5) is obtained with Jensen's inequality, Equation (6) from the factorized state transitions and Equation (7) by the assumption of independence between states.

In Equation (7), (a) is constant with the definition of q_θ and we ignore (c) in the discussion that follows. If we focus on (b) for an arbitrary t, we see that it can be rewritten according to Equation (8).

$$\begin{aligned}\text{(b)} &= \mathbb{E}_{q_\theta(X^t|X)}\left[\sum_{X^{t+1}\in\mathcal{S}} q_\theta(X^{t+1}|X^t, X)\log\frac{p_\theta(X^{t+1}|X^t)}{q_\theta(X^{t+1}|X^t, X)}\right] \\ &= -\mathbb{E}_{q_\theta(X^t|X)}\left[D_{\text{KL}}\left(q_\theta(X^{t+1}|X^t, X)\;||\;p_\theta(X^{t+1}|X^t)\right)\right] \end{aligned} \tag{8}$$

This is consistent with the loss term in Equation (4). However, contrary to the proof given by Zhang *et al*. [19], our expectation is rigorously derived. The problem with this expectation lies in the fact that $q_\theta(X^t \mid X)$ is intractable. In the formalism of diffusion models, an analogous derivation would be solved by using the reversal of the diffusion process. Unfortunately, in the case of infusion, there is no closed-form forward process from the complete shape X to X^t. Furthermore, the way samples are drawn when applying loss $\mathcal{L}_t$ does not follow the distribution over which the conditional expectation is taken, namely $q_\theta(X^t \mid X)$. Nevertheless, this loss remains intuitively consistent as it guides the optimized distribution to a target distribution, a scheme that has been proposed in previous works [11].

Degeneracy of the loss. As training progresses the joint distribution modeled by the learned transition kernel $p_\theta(o_\mathbf{p}, z_\mathbf{p})$ gets closer to $q_\theta(o_\mathbf{p}, z_\mathbf{p})$. However, $q_\theta(o_\mathbf{p}, z_\mathbf{p})$ depends directly on θ as it is a mixture of the latter and the exemplar as defined in Equation (2) and Equation (3). When training with a single exemplar, this distribution is very narrow and thus causes the loss to be unstable as training converges. This phenomenon is exacerbated considering that GCA uses a single network p_θ agnostic of the value of t to model state transitions. This is in stark contrast to diffusion models for which t is given as an additional parameter to the denoising network [7, 13]. In the case of GCA, this is however not conceivable as there is no fixed "forward process". In other words, the proposal distribution q_θ is both data-dependent and changes during training because it depends directly on the transition kernel p_θ that is being optimized. It is also worth noting that, for the same reason, there is no way to control variance in a principled manner in contrast to diffusion models for which the forward process is fixed and determined by the noise schedule. Furthermore, using Bernoulli transition kernels to model occupancy precludes simultaneous control over both mean and variance at the same time.

To mitigate these issues, we observed that it is crucial to train GCA with as few steps as necessary to reach the target distribution. As suggested by the original *Infusion* work [2], it is possible to sample more steps at inference than used during training. For our conditional generation task analogous to a form of completion, we thus choose $T = 5$ during training and $T = 7$ for generation.

S.2. Additional Implementation Details

All reported experiments were performed on a desktop with an AMD Ryzen 7 7700 CPU (8 cores), 32 GiB of RAM, and an NVIDIA GeForce RTX 4090 (24 GiB of VRAM). We use Linux kernel 6.5.0, Python 3.11.8, CUDA 11.8, and NVIDIA driver 550.

S.2.1. 3D Gaussian Splatting

Our implementation of 3D Gaussian Splatting is based on a standalone re-implementation of the *splatfacto* model in *nerfstudio* [14]. It uses *gsplat* v.0.1.11 for differentiable rendering routines. We use the default parameters of the *splatfacto* model with a few exceptions. To avoid large 3D Gaussians, we use the scale regularization of PhysGaussian [16] with a maximum Poisson ratio of 10.0. When pre-processing scenes, we clip the scale of 3D Gaussians so that their scale does not exceed twice the size of a voxel at the target voxel resolution r_t.

Our datasets were processed with COLMAP through *nerfstudio*. For challenging datasets, we used *Superpoint* descriptors [5] and *SuperGlue* [12] for matching. To extract DINO features, we relied on the implementation of EmerNeRF [17].

DINO features have high dimensions by default, typically 756, which would induce a significant memory and computational cost with a large number of Gaussians and be a more complicated signal to model in the generative task. Consequently, we reduce their dimension by applying a joint PCA across all the feature maps extracted from the training images. For simplicity and efficiency, we choose $d = 8$. We use DINOv1 features [3] as we observed artifacts when extracting DINOv2 features [10] as reported in previous works [17]. Contrary to colors, we distill DINO features as view-independent.

When applying affine transformation to 3D Gaussians during editing, we properly transform their mean, scale, and rotation components. Additionally, we rotate spherical harmonics as Gaussians often "bake" appearance into view-dependent effects.

S.2.2. Generative Cellular Automata

Our implementation of Generative Cellular Automata is based on the original codebase of Zhang *et al*. [18, 19] and uses MinkowskiEngine [4] for sparse tensor computations. However, we introduce a number of key modifications that we describe in this section.

We use the standard neighborhood over the generalized one $G_X(X^t)$ introduced in cGCA [19]. The reason is that our task can be seen as an inpainting task, where connectivity is ensured due to our choice of initialization: X^0 is obtained by downsampling the coarse conditioning voxels from resolution r_c to r_t. In practice, we use a neighborhood of size 2 w.r.t. L1 distance for our transition kernel. Our infusion schedule follows Zhang *et al*. [18, 19] and grows linearly from $\alpha_0 = 0.1$ to $\alpha_T = 0.25$. Similarly, σ_t follows $\sigma_t = e^{-1-0.01t}$.

To mitigate the degeneracy of the KL loss described in Appendix S.1.3 and considering that we can see the task of the GCA as an inpainting task starting from the upsampled coarse conditioning voxels, we use a low value for T. In practice, we choose $T = 5$ during training. At inference, we use $T = 7$ as it allows the GCA to self-correct itself for a couple of additional steps. We additionally perform one mode-seeking step as proposed by Zhang *et al*. [19].

Our model is a lightweight U-Net with two downsampling stages, each consisting of a single ResNet block with 16 base channels, doubling the number of channels at each resolution. Note that as we discussed previously, we don't condition the network on t. We use batch normalization instead of group normalization and ReLU activation instead of SiLU. We noticed that the implementation of Zhang *et al*. [19] does not input $o_\mathbf{p}$ to p_θ, which contradicts the factorization in Equation (1). We therefore reinstated $o_\mathbf{p}$ as an input to p_θ. We use *AdamW* [9] with a learning rate of 5×10^{-4} and weight decay of 1×10^{-6}. We train for only 10,000 iterations as convergence is reached quickly and longer training leads to overfitting due to the limited capacity of our model and data scarcity. We removed most implementation-level performance bottlenecks by batching every possible operation. Note that training and inference speed could certainly be improved with more advanced sparse tensor libraries and support for half-precision (e.g., fVDB [15]). We leave this for future works.

S.2.3. Patch Consistency Step

The dot product $\mathbf{F}_e \cdot \mathbf{F}_g$ in Equation 4 of the manuscript can be directly interpreted as a "masked" *cosine similarity* between the corresponding features. This holds because these features are normalized as presented in Section 3.2 of the manuscript (i.e., with values in $[-1, 1]$) and because we give unoccupied voxels within a patch a zero feature. In practice, we independently normalize each component of our mixture of features introduced in Section 3.2 of the manuscript, i.e., appearance and semantic features. Consequently, we implement d_{feat} as two independent distances that we simply average. The re-normalization term in Equation 4 of the manuscript guarantees that we don't prioritize patches with fewer intersecting voxels but consistent matching features.

S.3. Additional Ablations

S.3.1. Choice of Resolution

In Section 3.2 of the manuscript, we introduced a sparse volume of voxels with a specific resolution. In Figure S.3, we highlight the trade-offs between each resolution for conditioning and generation. The relative scale between the conditioning input and the target output is crucial for balancing fidelity and diversity Based on these observations, we choose $r_t = 64^3$ and $r_c = 16^3$ as a compromise, corresponding to a generation-to-conditioning ratio of 4.

S.3.2. Sparse Patch Consistency Step

In Figure S.4, we show the impact of each parameter of our sparse patch consistency operation. In the first row, we illustrate the impact of patch size $p = l^3$. A small patch size length (i.e., $l = 3$) matches local statistics and ignores structure. A large patch size leads to more intensive computations (especially for a large number of voxels) without significant improvements. Our patch consistency is evaluated by combining the occupancy d_{occ} and the feature distance d_{feat} as presented in Equation 3 of the manuscript. Biasing the total distance towards the occupancy term d_{occ} causes the algorithm to match dominant geometric statistics, which results in local flattening and repetitions. On the other hand, biasing it towards the feature term d_{feat} leads to severe artifacts. The expansion distance λ_{patch} defines the possible deviation range from the original generated voxels. The last row shows that a small λ_{patch} may severely confine the patch consistency step to the possibly erroneous output of GCA and lead to incomplete objects. On the other hand, a large value leads to spurious and uncontrolled expanded regions. Note that by default, we always use a patch size of length $l = 5$, 7 iterations, $w = 0.5$ for the weight between the occupancy distance d_{occ} and the feature distance d_{feat}, and a maximum expansion distance $\lambda_{\text{patch}} = 2.0$.

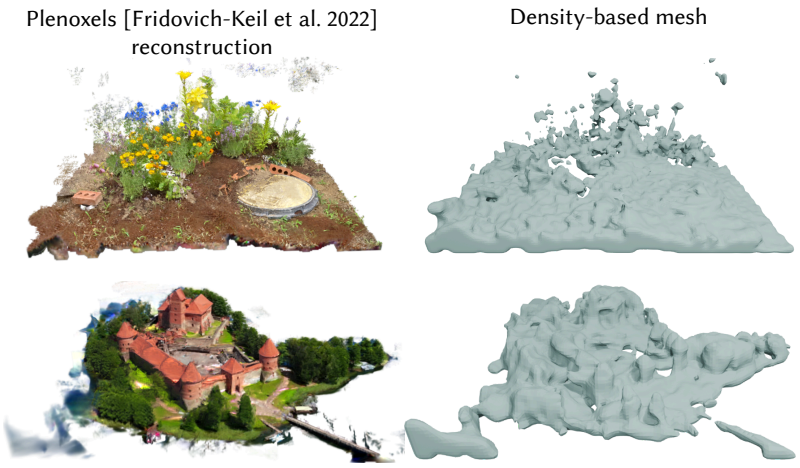

Figure S.1. Sin3DGen [8] reconstructs input scenes using Plenoxels [6]. Even after manual cropping, real-world scenes often contain significant artifacts that, when meshed from the radiance field density, yield an ill-defined SDF. This leads to unreliable generations and prevents proper conditioning, as Sin3DGen relies on the SDF to control generation (see Section 4 and Appendix B of [8]).

Treating appearance and geometry as fixed-size voxels that are only resampled from the exemplar naturally introduces discontinuities, as shown in Figure 15 of the manuscript. This can occur because the selection bounding box is not perfectly aligned with the structure of interest, the practical size of a voxel does not match the local structure of the exemplar, or because there is simply no way to bind two such regions of the scene. This phenomenon becomes particularly noticeable when training on very large distributions, as shown on the right of Figure 15 of the manuscript.

S.4. Key differences with Sin3DGen

In what follows we highlight the key differences between our patch-based consistency step introduced in Section 3.4 of the manuscript and the patch-based synthesis algorithm proposed by Sin3DGen [8].

Consistency step & controllable generation Our patch-based consistency step targets a different objective than Sin3DGen: it enforces consistency on the GCA's predictions as shown in Figure 4 of the manuscript. As a consequence, we use patch-based optimization at a single resolution (i.e., the target resolution r_t), whereas Sin3DGen generates patches hierarchically, and reproduces multi-scale patch statistics. By offloading the controllable generation task to GCA, we ensure explicit control over the generated scenes, even when they contain semantically structured objects. For editing, Sin3DGen [8] initializes coarser levels with a geometric proxy — analogous to our coarse conditioning voxels — and then synthesizes finer levels from that initialization. In Figure 8 and Section 4.3 of the manuscript, we show that trying to adapt this hierarchical patch-based synthesis strategy to our sparse formulation by ablating away GCA fails to respect the initial conditioning constraints, cannot handle semantically complex distributions and leads to repetitive patterns. Note that, as emphasized in the next paragraph, the two approaches behave differently due to the sparsity of our formulation.

Sparse and mesh-less formulation Even though Sin3DGen starts from Plenoxels [6], a sparse representation of a radiance field, it still synthesizes patches in a *dense* manner. This is key to propagate geometric changes during generation. More precisely, Sin3DGen first converts the density field into a mesh, extracts a dense SDF, and uses the corresponding truncated SDF as a feature to synthesize geometry. This approach works well for surface-like geometry, but fails for fuzzy regions that abound in real-world radiance fields and for which a mesh cannot be properly extracted from the radiance field as shown for the two scenes in Figure S.1. Furthermore, as it is prohibitively expensive, the process is made tractable by using *approximate* matching with a *Nearest-Neighbor Field* (NNF) introduced in *PatchMatch* [1]. On the other hand, with our method, geometry is already generated by the GCA and we are only interested in minor local refinements. In addition, we seek a formulation that can map directly to (a) the output of GCA, namely sparse voxels with features, and (b) the explicit nature of the 3D Gaussians we use to model appearance: we do not model a radiance field that can easily be converted to signed distance function values. Note also that by keeping a sparse representation, we can match patches exhaustively without approximations. This choice entails designing a proper distance function and a way to update geometry locally (voting mechanism).

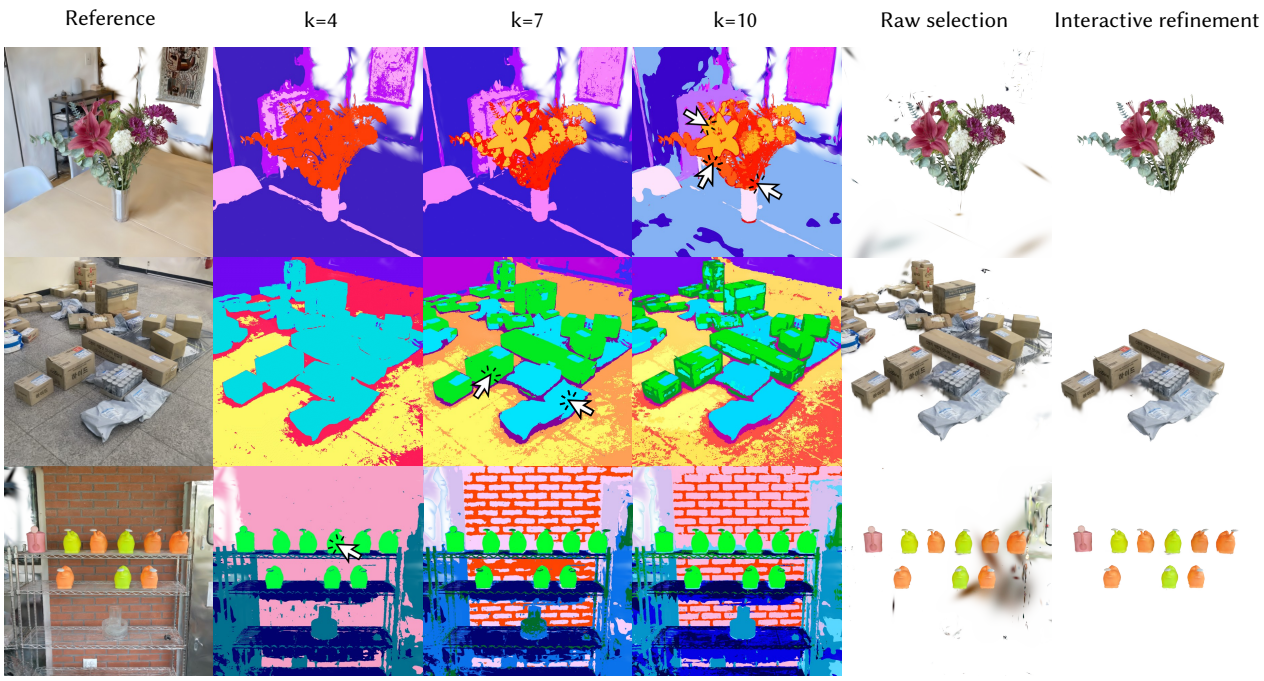

Figure S.2. In addition to view-dependent color, we distill low-dimensional view-independent features $\mathbf{f}$, namely PCA-ed DINO features. When clustered and rendered, these features provide semantic segments that can be used to select regions within the scene. In columns 2 to 4, we show the segments that are instantly exposed to the user for different values of k. Upon selection, the corresponding 3D Gaussians can be isolated (column 5). In column 6, we show that additional refinements can be performed using an interactive 3D selection tool shown at the bottom of Figure 5.a of the manuscript and in the supplemental video.

Distance choice & voting mechanism In essence, our matching distance is inspired by Sin3DGen as it balances geometry and feature consistency. However, we represent occupancy explicitly as a binary variable $\mathbf{O}_i$ and not a signed distance value, and we compare features using cosine similarity instead of L2 norm. As our features are renormalized (i.e., unit norm) and empty voxels in a patch are given zero features, this allows us to only compare features for which occupancy is matched. Similarly to Sin3DGen, our features are based on PCA-ed spherical harmonics but we leverage additional guidance from semantic features, namely PCA-ed DINO features. As we rely on a sparse representation, we cannot update and propagate geometry through a continuous signed distance field. To this extent, our voting mechanism allows us to *refine* geometry locally.

S.5. Additional figures

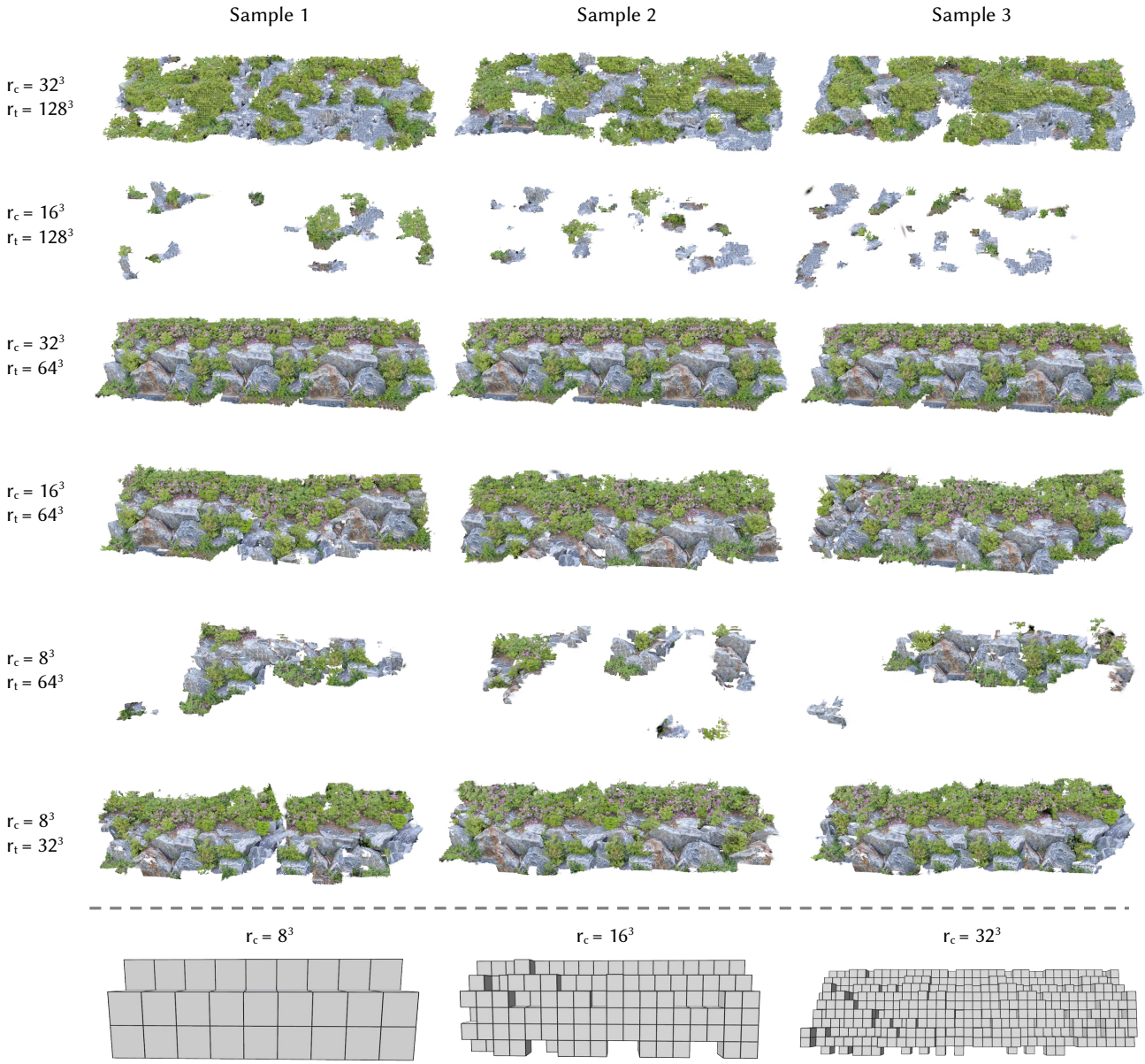

Figure S.3. Illustration of the impact of the target resolution r_t and conditioning resolution r_c. The first row shows that generating at a very fine resolution (i.e., $r_t = 128^3$) causes the GCA to focus on higher-frequency details and miss structure. Conversely, a coarse target resolution (i.e., $r_t = 32^3$) results in "blocky" outputs. The relative scales between the conditioning and the target are crucial to balance fidelity and diversity. If the difference is too high, GCA fails to recover the shape. This is due to our choice of neighborhood and the conditioning mechanism introduced in Section 3.3.2 of the manuscript. Conversely, conditioning at a resolution too close to the target one (middle row) results in complete overfitting on the exemplar.

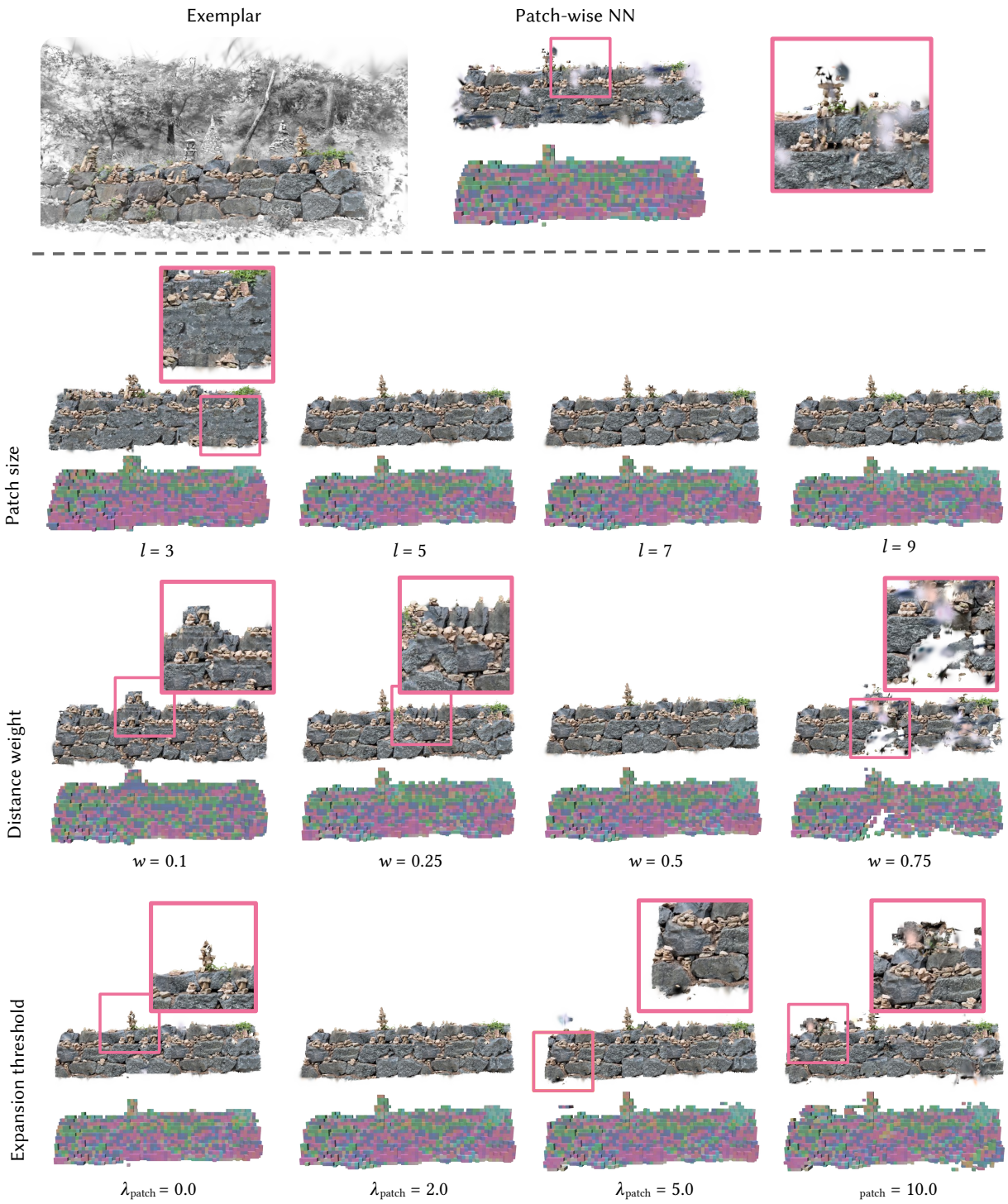

Figure S.4. Impact of the different parameters of our sparse patch consistency algorithm. For each parameter, we start from our default configuration, namely $l = 5$, $w = 0.5$, and $\lambda_{\text{patch}} = 2$, and vary the parameter of interest. Patch size side-length l trades off local statistics for larger ones at the cost of increased computations. Distance weight w balances occupancy matching with patch-wise feature matching. Maximum expansion distance λ_{patch} enables geometric corrections but leads to artifacts when set too high.

References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28(3):24, 2009. 5

[2] F. Bordes, S. Honari, and P. Vincent. Learning to generate samples from noise through infusion training. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. 2, 3

[3] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021. 3

[4] C. Choy, J. Gwak, and S. Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019. 2, 4

[5] D. DeTone, T. Malisiewicz, and A. Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 224–236, 2018. 3

[6] S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022. 5

[7] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. 3

[8] W. Li, X. Chen, J. Wang, and B. Chen. Patch-based 3d natural scene generation from a single example. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16762–16772, 2023. 5

[9] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 4

[10] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023. 3

[11] D. Ritchie, A. Thomas, P. Hanrahan, and N. Goodman. Neurally-guided procedural models: Amortized inference for procedural graphics programs using neural networks. *Advances in neural information processing systems*, 29, 2016. 3

[12] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4938–4947, 2020. 3

[13] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015. 3

[14] M. Tancik, E. Weber, E. Ng, R. Li, B. Yi, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja, et al. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–12, 2023. 3

[15] F. Williams, J. Huang, J. Swartz, G. Klar, V. Thakkar, M. Cong, X. Ren, R. Li, C. Fuji-Tsang, S. Fidler, et al. fvdb: A deep-learning framework for sparse, large scale, and high performance spatial intelligence. *ACM Transactions on Graphics (TOG)*, 43(4):1–15, 2024. 4

[16] T. Xie, Z. Zong, Y. Qiu, X. Li, Y. Feng, Y. Yang, and C. Jiang. Physgaussian: Physics-integrated 3d gaussians for generative dynamics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4389–4398, 2024. 3

[17] J. Yang, B. Ivanovic, O. Litany, X. Weng, S. W. Kim, B. Li, T. Che, D. Xu, S. Fidler, M. Pavone, et al. Emernerf: Emergent spatial-temporal scene decomposition via self-supervision. *arXiv preprint arXiv:2311.02077*, 2023. 3

[18] D. Zhang, C. Choi, J. Kim, and Y. M. Kim. Learning to generate 3d shapes with generative cellular automata. *arXiv preprint arXiv:2103.04130*, 2021. 1, 4

[19] D. Zhang, C. Choi, I. Park, and Y. M. Kim. Probabilistic implicit scene completion. *arXiv preprint arXiv:2204.01264*, 2022. 1, 2, 3, 4