DreamCraft: Text-Guided Generation of Functional 3D Environments in Minecraft

Anonymous Author(s)
Affiliation
Address
email

Abstract

Procedural Content Generation (PCG) algorithms enable the automatic generation of complex and diverse environments. However, they don't provide high-level control over the generated content and typically require domain expertise. In contrast, text-to-3D methods allow users to specify desired characteristics in natural language, offering a high amount of flexibility and expressivity. But unlike PCG, such approaches cannot guarantee functionality, which is crucial for certain applications like game design. In this paper, we present a method for generating functional 3D environments from free-form text prompts in the open-world game Minecraft. Our method, *DreamCraft*, trains a quantized NeRF to produce an environment layout that, when viewed in-game, matches a given text description. We find that DreamCraft produces more aligned in-game artifacts than a baseline that post-processes the output of an unconstrained NeRF. Thanks to the quantized representation of the environment, functional constraints can be integrated using specialized loss terms. We show how this can be leveraged to generate 3D structures that match a target distribution or obey certain adjacency rules over the block types. DreamCraft inherits a high degree of expressivity and controllability from the NeRF, while still being able to incorporate functional constraints through domain-specific objectives.

1 Introduction

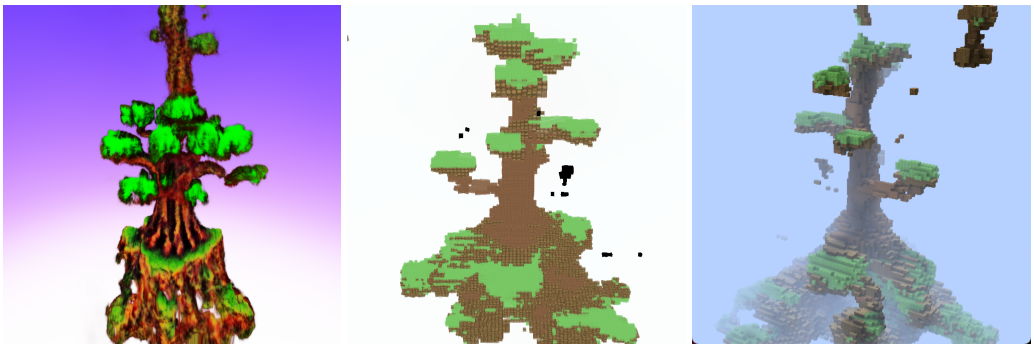
Procedural Content Generation (PCG) refers to a class of algorithms that can automatically create content such as video game levels [61, 26, 53, 33, 8, 40], 2D or 3D visual assets [80, 40, 5, 48, 38, 67], game rules or mechanics [70, 49, 76, 16, 84], or reinforcement learning (RL) environments [31, 30, 53, 34, 9, 57, 73, 28, 14, 3, 74, 29, 50]. PCG allows for compression of information [69, 76], increased replayability via endless variation [81, 66, 4], expression of particular aesthetics [37, 1, 6, 21], and reduction of human labour otherwise required to manually produce artifacts [60, 13, 10, 62]. In addition, it produces functional content by design since it directly leverages a generator. While PCG can automatically generate complex and diverse environments, the generations are not open-ended and cannot be controlled using free-form specifications, being constrained by the generator's expressivity. In general, PCG also requires domain knowledge since each algorithm is specific to its use case.

In contrast, recent generative models have shown impressive abilities of generating diverse images, videos, or 3D scenes from text prompts describing the desired output in natural language [54, 51, 63]. These advances allow users to create high-quality content even if they are not domain experts. While these models can produce controllable and open-ended generations, the created content is not guaranteed to be functional. Functionality is particularly important for certain applications such as game design or the creation of RL environments.

In this paper, we propose a new method for generating functional 3D environments from free-form text prompts in the open-world game Minecraft. Our method, *DreamCraft* trains a quantized NeRF

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

(a) *large medieval ship*



Unconstrained NeRF DreamCraft Neural Render DreamCraft In-Game

(b) *old wizards tree mansion series 2 build 1read description*

Figure 1: **Generations** given Planet Minecraft asset descriptions from an Unconstrained NeRF (left), DreamCraft visualized via neural rendering (middle), and DreamCraft visualized in-game (right).

to produce an environment layout that, when viewed in-game, matches a given text description. Experimenting with various quantization schemes, we find that using soft density blocks or annealing them from continuous to discrete is crucial for stable learning, while u sing discrete block types leads to the most recognizable structures. We evaluate the fidelity of our approach in matching generated artifacts to descriptions of both generic and domain-specific scenes and objects. We find that DreamCraft produces in-game artifacts that align with inputs more consistently than a baseline that post-processes the output of an unconstrained NeRF.

Thanks to its quantized representation of the game world, DreamCraft can jointly optimize loss terms that enforce local functional constraints on patterns of blocks. We show how this can be instantiated to, for example, generate 3D structures that match a target distribution no or obey certain adjacency rules over the block types. By inheriting a high degree of expressivity and controllability from the NeRF, while still being able to incorporate functional constraints through domain-specific objectives, DreamCraft combines the strengths of both PCG and generative AI approaches, representing a first step towards democratizing flexible yet functional content creation. Our method has potential applications in the development of AI assistants for game design, as well as in the production of diverse and controllable environments for training and evaluating RL agents.

To summarize, our paper makes the following contributions: (i) introduces a new method for training a quantized NeRF to produce 3D structures that match a given textual description using a set of discrete Minecraft blocks, (ii) studies different quantization schemes such as whether to use discrete or continuous block densities and types, (iii) shows that the quantized NeRF produces more accurate Minecraft artifacts than an unconstrained NeRF, (iv) and demonstrates how to incorporate functional constraints such as obeying certain target distributions or adjacency rules.

2 Related Work

Procedural Content Generation (PCG) is becoming increasingly more popular for training and evaluating robust RL agents that can generalize across a wide range of settings [31, 30, 34, 9, 57,

28, 73, 74]. Generative models like ours provide a way of biasing the environment generations towards human-relevant ones, thus enabling to search more efficiently through vast environment spaces. Existing works indicate that environment generation can be controlled via computable metrics [12, 11, 33, 29, 18, 58, 59, 47, 60]. Novel environments can also be generated by learning on human datasets [64, 23, 24, 40, 42, 70], sometimes with additional functional constraints or post-processing [22, 83, 36, 77, 32]. More recently, [75, 68] use large language models to generate Sokoban and Mario 2D levels. However, our work is first to show how multi-modal models can be leveraged to guide the generation of 3D game environments. One of the most popular PCG algorithms is wave function collapse [19] which generates structurally consistent content from a single sample, such that its output matches tile-frequency and adjacency constraints. In this paper, we show how such constraints can be used in conjunction with text-guidance to generate environments where both high-level and low-level aspects can be controlled.

Text-to-3D Generation Our work builds upon the many recent advances in text-to-3D generation [51, 45, 7]. For example, DVGO [71] is a supervised NeRF method that, instead of training an MLP, directly optimizes a voxel grid over the 3D space. Similarly, PureCLIPNeRF [35] uses a CLIP loss to guide a NeRF using both direct and implicit voxel grids. Our approach, DreamCraft, resembles an implicit voxel grid approach, in that it uses MLPs to parameterize activations over discrete grids. But instead of outputting continuous RGB and density values and interpolating between nearest grid vertices (to determine activation at a given point during ray tracing), our approach uses MLPs to produce predictions over block types, and considers only the single nearest grid vertex during ray sampling (to determine within which block a given point resides).

Minecraft Environment Generation Several prior works have sought to generate Minecraft environments using both supervised and self-supervised methods. [2] use a 3D GAN [15] architecture to generate arbitrarily sized world snippets from a single example. Similarly, [25] use an unsupervised neural rendering framework to generate phtorealistic images of large 3D Minecraft structures. To assist Minecraft players, [44] introduces a tool for interactive evolution using both a 3D generative model to generate the structure design and an encoding model for applying Minecraft-specific textures to the structure's voxels. [67] has recently shown that neural cellular automata (NCAs) can be used to grow complex 3D Minecraft entities made out of thousands of blocks such as castles, apartment blocks, and trees. Other works have leveraged search-based methods [82], evolutionary algorithms [43, 65], or even reinforcement learning [29] approaches to generate Minecraft structures. The game has also been a testbed for PCG algorithms [55, 56], open-endedness [17], and artificial life [67]. However, our work is first to generate functional Minecraft environments directly from text prompts, enhancing the high-level controllability of the creation process.

3 DreamCraft: Text-Guided Minecraft Environment Generation

3.1 Quantized NeRFs for Environment Generation

In this section, we introduce **DreamCraft**, a quantized NeRF which learns to arrange in-game assets during training. Our NeRF implementation builds upon "stable-dreamfusion", an open-source implementation of DreamFusion [72] that uses Stable Diffusion [54] for text-guidance. Note that our approach is agnostic to the text-to-3D model used, so it can be applied to any other pretrained text-to-3D model and thus benefit from future advances in that field. Also note that we do not use any additional or domain-specific data to train our model apart from the block textures in Table 4.

From Continuous Points to Discrete Blocks. As in PureCLIPNeRF, we use MLPs to predict continuous activations over a grid by feeding them X, Y, Z coordinates which are encoded using a series of sine waves, as in the original NeRF [45]. The output of a NeRF is taken as a block grid in which each vertex represents the center of a block, and is a probability distribution over block types. We take gumbel softmax [27] over these predictions to yield a discrete grid of blocks. We denote the *Minecraft block grid* $B \in \{0,1\}^{N\times N\times N}$, of width N, comprising of M different non-air block types. We maintain a separate density grid indicating the transparency of each block, yielding an *air block grid* $A \in \{0,1\}^{M\times N\times N\times N}$.

From 2D Textures to 3D Structures. We now render the onehot block grid approximately as it would appear in-game. To achieve this, when the NeRF is initialized, each block type is associated with one or several in-game textures, which we arrange in a 3D voxel grid, using 2D textures as faces

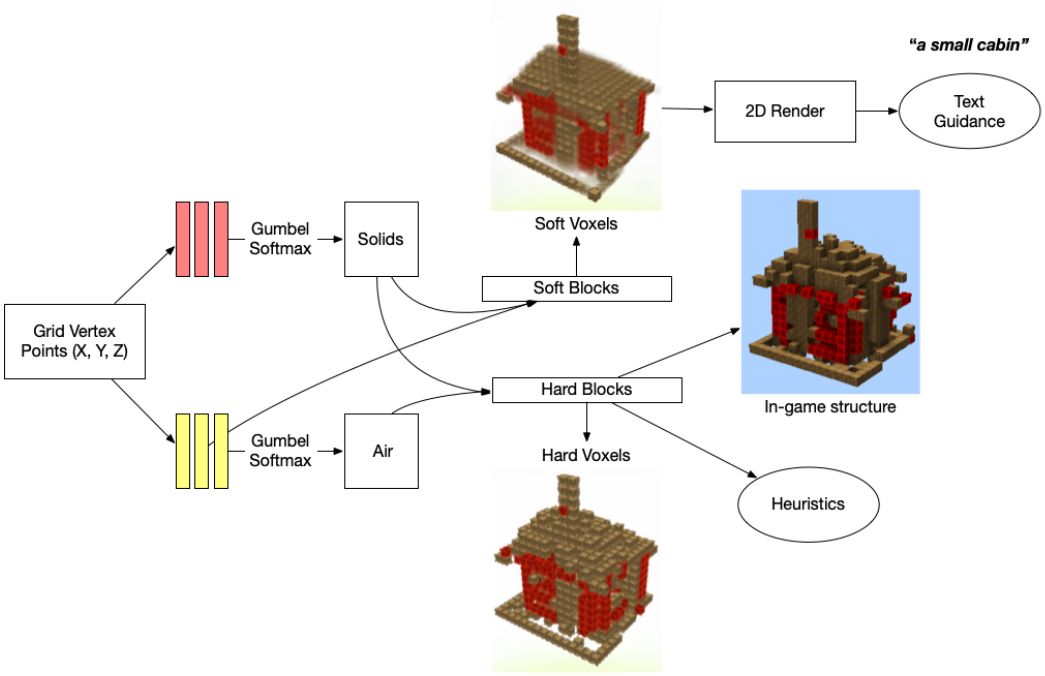


Figure 2: **DreamCraft** produces a distribution of discrete blocks over a grid. Unquantized or "soft" versions of this distribution can be visualized for text-image guidance, while a fully discrete representation can be used to determine functional constraints, and exported to an in-game structure.

of a 3D cube (see Figure 3). In Minecraft, this results in a $16 \times 16 \times 16$-sized voxel grid. During the forward pass, the discrete block grid is projected into a higher-resolution voxel grid made up of block-specific voxel grids. When sampling points along a ray during rendering, each point takes the color and density values of the voxel in which it falls, avoiding interpolation between cells to mimic the sharp, pixelated appearance of textures in game.

Given a vertex $\mathbf{x} \in \mathbb{R}^3$ on the block grid, we compute a discrete block type, first generating a prediction $\mathbf{b}_{\text{soft}} \in \mathbb{R}^M$ over M block types, then obtaining a onehot vector $\mathbf{b}_{\text{hard}} \in \{e_1, e_2, \ldots, e_M\}$ using the gumbel softmax function:

$$\mathbf{b}_{\text{soft}} = \text{MLP}\left(\mathbf{x}; \theta_B\right) \qquad \mathbf{b}_{\text{hard}} = \text{gumbel\_softmax}\left(\mathbf{b}_{\text{soft}}\right),$$

where θ_B denotes the parameters of the solid block type MLP.

Again given the block grid vertex $\mathbf{x}$, we compute soft and hard density values, where high values correspond to solid blocks and low values approaching 0 correspond to air.

Soft density is computed as in previous works, passing the output of an MLP (parameterized by θ_A) through an exponential activation function[1]:

$$\sigma_{\text{soft}} = \exp\left(\text{MLP}(\mathbf{x}; \theta_A)\right),$$

where θ_A denotes the parameters of the air block MLP. To quantize the density grid (effectively computing the presence/absence of air blocks), we first use the soft density values to derive the respective probability of an air/solid block appearing at x:

$$\sigma'_{\text{soft}} = \sigma_{\text{soft}} - 10 \qquad p_{\text{air}} = -\sigma'_{\text{soft}} \qquad p_{\text{solid}} = \sigma'_{\text{soft}}$$

We then use the gumbel softmax function to discretize these predictions, obtaining a density value of 0 in case of air, and 1 in case of a solid block:

$$\sigma_{\text{hard}} = \sum^{2} \text{gumbel\_softmax}\left([p_{\text{air}}, p_{\text{solid}}]\right)$$

[1]During the backward pass, the exponential function is clipped to avoid exploding gradients: $\sigma_{\text{soft}} = \exp\left(\text{clamp}(y, 15)\right)$.

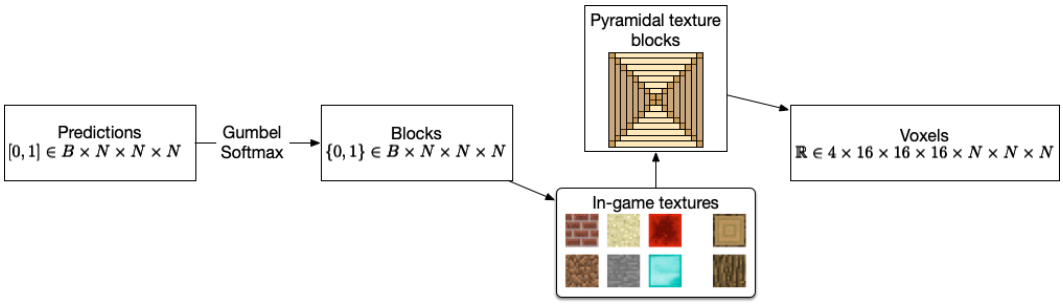


Figure 3: **From 2D textures to 3D structures.** Once discretized, block types are mapped to voxel grids corresponding to the appearance of objects in-game. Here, we assemble each block into a 3D voxel grid using 2D textures from Minecraft. We approximate solid blocks by repeating the surface texture in the space inside the block.

To obtain a final discrete grid of Minecraft blocks $\mathbf{c}$, including air and solids, we mask the solid block grid $\mathbf{b}_{\text{hard}}$ by the density grid σ_{hard}, placing air blocks wherever density is 0.

We use only solid Minecraft blocks. When sampling RGB and density values along a ray point $u \in G$, the density at any point closest to grid vertex $\mathbf{x}$ is multiplied by a solidity constant $c = 100$, $\sigma_{\text{hard}(u)} = c \times x_u$, width $x_u = \arg\min_x x - u$. We use the "hard" gumbel softmax function, forcing the result to be a onehot vector.

As in DreamFusion, we use a separate, shallow background MLP, which takes as input a viewing angle, to model colour at the "end" of each ray, allowing the model to learn a low-resolution background texture (effectively projected onto the inside of a sphere).

3.2 Functional Constraints

Distributional Constraints. The discrete block grid resulting from the quantized NeRF allows us to optimize DreamCraft to produce a level satisfying a target distribution of block types, producing text-guided objects comprising of particular block mixtures. The user sets a target proportion for each block type, and we apply a loss equal to the difference between this target and the actual proportion of (non-air) grid cells in the NeRF's output (after quantization), which we compute by taking the sum over the relevant channel of the discrete onehot block grid and dividing by the size of the grid. Formally, we define the distributional loss as

$$L_D = \sum_{t \in T} |G(t) - P(t)|,$$

where t is a block type among the set of blocks T, $G(t)$ is the target number of occurrences of this block as specified by the user, and $P(t)$ is the number of actual occurrence in the quantized block grid $\mathbf{c}$.

Adjacency Constraints. We can also add a loss term corresponding to a penalty or reward incurred whenever a particular configuration of blocks appear in the generated structure. To this end, we construct a convolutional layer that outputs 1 over any matching patch of blocks, and 0 everywhere else, then sum the result. We multiply this sum by some user-specified loss coefficient (negative when the pattern is desired, positive when prohibited). Suppose the user wants to apply a loss/reward of l_p to the pattern of blocks $b_0, b_1, \ldots, b_{j_p}$ occupying a patch of size K^3 (with the number of blocks of interest $j_p \leq K^3$), where each block b_i has relative coordinates x_i, y_i, z_i in the patch. We construct a 3D convolutional weight W_p consisting of the onehot vectors e_i corresponding to each block type, and placing them at position x_i, y_i, z_i in the weight matrix. We apply the resulting convolutional layer, conv_{W_p} to the quantized block grid, subtract $j_p - 1$ from the output, and apply a ReLU activation to obtain the binary pattern-activation tensor. Formally,

$$L_P = \sum_{p \in P} w_p \sum_{i}^{K^3} \left(\text{ReLU}\left(\text{conv}_{W_p}(\mathbf{c}) - j_p + 1\right)\right)_i,$$

where an inner sum is taken over elements i in the binary activation tensor for pattern p.

	COCO		Planet Minecraft	
Model	CLIP ViT-B/16	CLIP ViT-B/32	CLIP ViT B/16	CLiP ViT B/32
Unconstrained NeRF	**61.44**	**66.67**	**25.17**	**31.29**
DreamCraft	19.74	21.05	11.56	17.01
ratio	0.32	0.32	0.46	0.54

Table 1: **Fidelity of Neural Renders on COCO and Planet Minecraft.** Quantized generations produced are comparable to unquantized generations from an Unconstrained NeRF, which can be considered an upper bound. DreamCraft's relative performance increases when moving to a set of domain-relevant prompts.

Model	CLIP ViT-B/16	CLIP ViT-B/32
Unconstrained NeRF	2.72	2.72
DreamCraft	**5.44**	**6.12**

Table 2: **Fidelity of In-Game Renders on Planet Minecraft.** DreamCraft, which learns to use discrete blocks during training, outperforms a baseline whose output is discretized only after training.

4 Experiments

Baselines. We compare the performance of DreamCraft, our quantized NeRF which learns to arrange in-game assets *during training*, to **Unconstrained NeRF**, a baseline that maps the continuous outputs to game assets *after training*. Unconstrained NeRF trains a DreamFusion model and then maps its output to Minecraft blocks using nearest neighbors. The nearest RGB value to each Minecraft block is determined by taking the average color over each of the (normally repeated) 16×16 textures covering each of its 6 faces. We define a width-N grid over the 3D output space of the unconstrained model and query the RGB and density values at the center of each cell in the grid. We calculate the L_2 distance between each centerpoint and average Minecraft block color, mapping each cell to the closest block. We then select a density threshold $s = 10$, and place air blocks wherever $\sigma < s$.

Ablations. We study different quantization schemes in order to understand what is the best way to map the continuous outputs of the air and solid block MLPs into discrete grids of Minecraft blocks. The output of either MLP can be passed through the $\mathrm{gumbel\_softmax}$ function to produce a discrete grid of air or solid blocks (see Figure 2). If these values are not discretized *i.e.,* $\mathbf{b}_{\mathrm{soft}}$ or σ_{soft} are used instead of their "hard" counterparts, then the resulting voxel grid can include solid blocks interpolated with air blocks (*i.e.,* semi-transparent) or with one another (*i.e.,* multi-texture). We also experiment with linearly annealing these values from their soft to hard counterparts over the course of training.

Evaluation Datasets. We evaluate our method on both generic and domain-specific text prompts, using the **COCO** [39] and **Planet Minecraft** [41] datasets, respectively. For COCO, we use the same 153 prompts as in prior text-to-3D works [46, 51, 35]. For Planet Minecraft, we take the names of the top 150 most downloaded assets uploaded by users in 2016 under the "Maps" category. Some examples include "a desperate and lonely wizards tower pmc chunk challenge entry lore", "mario kartgba bowsers castle 2", and "icarly set and nickelodeon studio". See Section F for the full list.

Evaluation Metrics. To quantitatively evaluate the performance of our model, we measure its fidelity using **R-precision**. More specifically, we query other pre-trained joint text-image encoders, namely CLIP ViT-B/16 and CLIP ViT-B/32 [52], and test whether they can recognize the caption responsible for a given NeRF rendering from a set of distractors (other, randomly selected captions from the dataset). For each caption, we repeat the process for 5 different test images and average the result.

4.1 Quality of the Generations

In Figures 1 and 8, we can see that, using only Minecraft blocks, our model produces structures that are visually similar to those of the Unconstrained NeRF, for both generic and domain-specific text prompts. Note that the Unconstrained NeRF model is a strong upper bound because it has a continuous and thus much larger output space than our discretized DreamCraft model (see Table 1).

block type	block density	CLIP ViT-B/16 RGB	CLIP ViT-B/16 depth	CLIP ViT-B/32 RGB	CLIP ViT-B/32 depth
anneal	anneal	7.73	5.07	9.07	5.73
	hard	2.67	1.20	5.60	2.00
	soft	7.33	7.33	10.40	5.20
hard	anneal	8.53	6.00	8.40	7.33
	hard	2.27	1.33	2.80	1.87
	soft	**10.40**	**8.93**	**13.20**	**8.40**
soft	anneal	8.40	3.60	10.00	4.80
	hard	4.00	0.80	7.60	2.80
	soft	7.87	6.53	11.07	5.47

Table 3: **Fidelity of Different Quantization Schemes for Planet Minecraft.**. Using a hard block type and soft block density achieves the best performance.

Fidelity of Neural Renders on COCO and Planet Minecraft. Quantized generations produced are comparable to unquantized generations from an Unconstrained NeRF, which can be considered an upper bound. DreamCraft's relative performance increases when moving to a set of domain-relevant prompts.

In Table 1, we show the fidelity of DreamCraft and the Unconstraine NeRF baselines on COCO and Planet Minecraft. Note that here, the generations are evaluated using the neural rather than in-game renders. As expected, limiting the NeRF's output to a specific set of discretely assembled blocks drastically reduces its space of generations. This is reflected in DreamCraft's lower fidelity with respect to the Unconstrained NeRF, when generating objects from both the COCO dataset and Planet Minecraft datasets. However, the performance gap between DreamCraft and the Unconstrained NeRF is reduced when moving from the generic COCO dataset to the domain-specific Planet Minecraft dataset. This suggests that despite its restricted output space, DreamCraft is particularly capable of generating high-quality structures when the input prompts are relevant to the domain at hand.

In Table 6, we evaluate the R-precision using 2D captures of generated Minecraft block layouts in the game engine itself. Note that here, the generations are evaluated using the in-game renders rather than the neural renders. In this case, the fidelity of the Unconstrained NeRF is lower than that of DreamCraft for both COCO and Planet Minecraft. This indicates that post-processing the output of a NeRF in order to discretize it using nearest-neighbor leads to worse results than learning to use discrete blocks during the generation process. In Figure 9, we see that mapping a discrete set of grid vertices to nearest neighbor block types via average color leads to sub-optimal results that are particularly bad at maintaining consistency in terms of texture and color. This result demonstrates the difficulty of translating unconstrained generations to a constrained repertoire of domain-specific assets. We conclude that by incorporating these game assets in the learning process, we can generate more faithful in-game structures.

4.2 Quantization Schemes

In Table 3, we compare the effect of applying soft, hard, or annealed quantization schemes to solid and air blocks. Maintaining *soft air blocks* (continuous-valued block transparency), in combination with *hard solid blocks* (discrete block types), leads to the highest R-precision at test time. This suggests that learning the topology (in contrast to the color/texture) of a generated structure is a sensitive process in which relaxing the quantization scheme (and presumably simplifying the loss landscape) is crucial.

Forcing block density to be discrete throughout training leads to poor performance (as in the first row of Figure 4), with the poorest performance coming from models in which both block type and density are fully discrete. This may be owing to noisier learning dynamics resulting from the quantized output space of the model.

Conversely, using soft block density can lead to situations in which apparently solid surfaces are emulated by layering a number of semi-transperent blocks. At test time, when rendering the fully

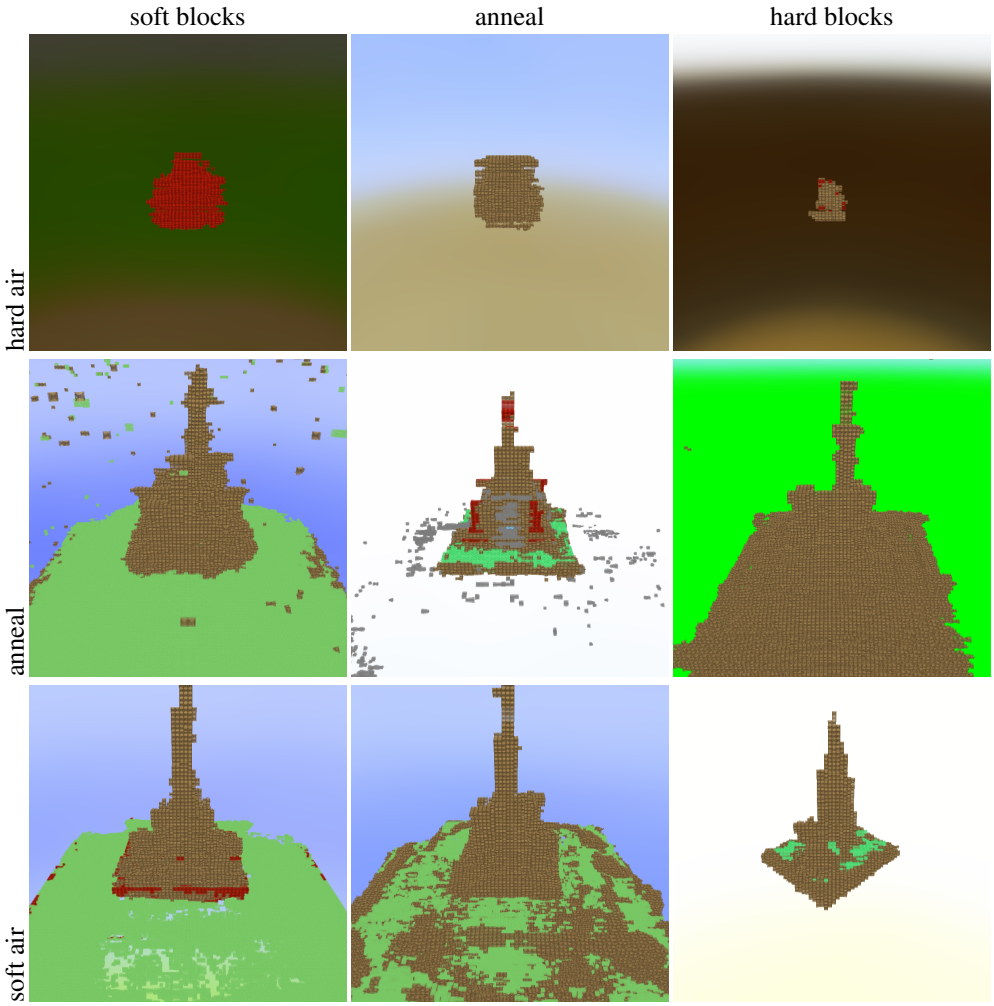


Figure 4: **Qualitative Effect of Different Quantization Schemes** for *"dwarven entrance"*. The best results obtained using hard block types and soft air blocks. Meaningful representations are learned only when the discreteness of air blocks (or by analogy, the density or topology of the generated structure) is relaxed or annealed throughout training (bottom two rows).

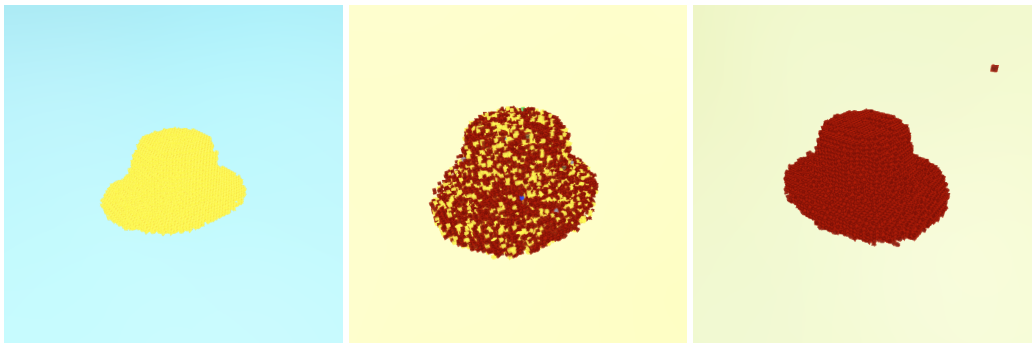
.

discrete block grid, such surfaces can suddenly be culled from the image, as none of the individual blocks of which they consist are enough to result in a "solid" binary output after quantization.

4.3 Functional Constraints

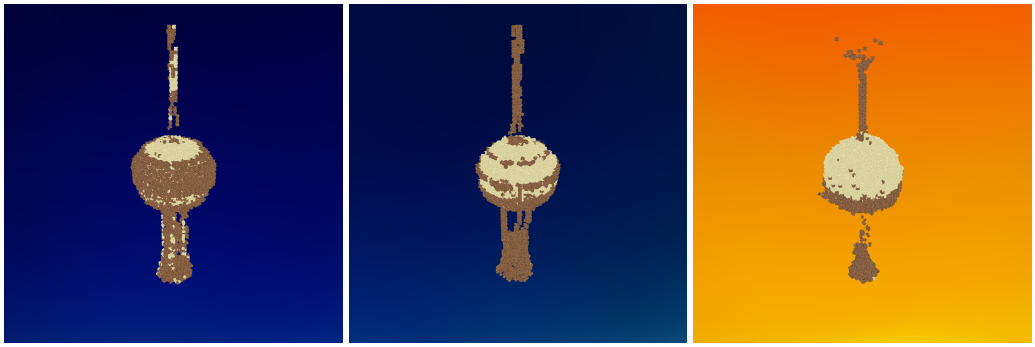
In Figure 5, we illustrate the effect of adding distributional constraints to a prompt asking for "a stylish hat". We can see that the model produces a similar structure using either entirely gold, redstone, or an even split of both when adding the distributional loss term.

In Figure 6, we illustrate the effect of adding an adjacency constraint prohibiting sand from "floating", i.e. being placed directly above an air tile, and ask for "space needle accurate" (a Planet Minecraft prompt). As the weight of the adjacency loss term is increased, the use of sand blocks becomes restricted to the central "bulb" of the tower, where it is more likely to be supported by a the circular base of dirt blocks. When the adjacency loss is weighted less heavily, sand is often incorporated into the underside of the bulb (where it is unsupported and will fall to the ground in-game).



$n(\text{gold}) = 1.0$ $n(\text{redstone}) = n(\text{gold})$ $n(\text{redstone}) = 1.0$

Figure 5: **Distributional Constraints.** *"a stylish hat"* jointly optimized to satisfy a given target distribution over the block types.



$w = 0.25$ $w = 1.75$ $w = 2$

Figure 6: **Adjacency Constraints.** *"space needle accurate"* with a penalty for floating sand, weighted more heavily from left to right. The block set is limited to sand (light tan) and dirt (brown).

These experiments demonstrate that functional constraints can be easily integrated with our text-to-3D model to create controllable Minecraft structures that obey both high-level and low-level user specifications and can be rendered in-game.

5 Conclusion

In this work, we develop a new approach for generating functional game environments in Minecraft from free-form text descriptions. DreamCraft quantizes the output of a text-to-3D NeRF to predict discrete block types which are then mapped to game assets (*i.e.,* voxel-grids corresponding to in-game blocks). This allows the NeRF to use the game assets to represent the content described by a text prompt. We demonstrate that our approach has higher fidelity to the text prompt than a baseline that discretizes the output of an unconstrained NeRF *after* learning. DreamCraft is, to our knowledge, the first procedural generator capable of generating diverse, functional, and controllable game environments directly from free-form text. Since our model can adapt to the unique appearance of user-supplied modular game assets to produce environments with high-level aesthetic properties, it may be particularly useful for game designers working in new domains that don't yet have have large datasets of game layouts.

One limitation of DreamCraft is that it takes a few hours to generate a single structure. However, it could benefit from recent and future speed improvements in NeRFs [79, 20, 71, 83]. Another promising direction for future work is to model lightning and shadow, in addition to color and density, which could be achieved using an auxiliary model to model the in-game lightning effects.

References

[1] Alberto Alvarez, Steve Dahlskog, Jose Font, Johan Holmberg, and Simon Johansson. Assessing aesthetic criteria in the evolutionary dungeon designer. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, pages 1–4, 2018.

[2] Maren Awiszus, Frederik Schubert, and Bodo Rosenhahn. World-gan: a generative model for minecraft worlds. In *2021 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2021.

[3] Philip Bontrager and Julian Togelius. Learning to generate levels from nothing. In *2021 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2021.

[4] Nathan Brewer. Computerized dungeons and randomly generated worlds: From rogue to minecraft [scanning our past]. *Proceedings of the IEEE*, 105(5):970–977, 2017.

[5] Michele Brocchini, Marco Mameli, Emanuele Balloni, Laura Della Sciucca, Luca Rossi, Marina Paolanti, Emanuele Frontoni, and Primo Zingaretti. Monster: A deep learning-based system for the automatic generation of gaming assets. In *Image Analysis and Processing. ICIAP 2022 Workshops: ICIAP International Workshops, Lecce, Italy, May 23–27, 2022, Revised Selected Papers, Part I*, pages 280–290. Springer, 2022.

[6] Alessandro Canossa and Gillian Smith. Towards a procedural evaluation technique: Metrics for level design. In *The 10th International Conference on the Foundations of Digital Games*, page 8. sn, 2015.

[7] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*, pages 333–350. Springer, 2022.

[8] Steve Dahlskog and Julian Togelius. Procedural content generation using patterns as objectives. In *Applications of Evolutionary Computation: 17th European Conference, EvoApplications 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers 17*, pages 325–336. Springer, 2014.

[9] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

[10] Robert Ota Dieterich. *Using Proof-Of-Concept Feedback to Explore the Relationship Between Artists and Procedural Content Generation in Computer Game Development Tools*. PhD thesis, 2017.

[11] Sam Earle, Maria Edwards, Ahmed Khalifa, Philip Bontrager, and Julian Togelius. Learning controllable content generators. In *2021 IEEE Conference on Games (CoG)*, pages 1–9. IEEE, 2021.

[12] Sam Earle, Justin Snider, Matthew C Fontaine, Stefanos Nikolaidis, and Julian Togelius. Illuminating diverse neural cellular automata for level generation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 68–76, 2022.

[13] Tianhan Gao, Jin Zhang, and Qingwei Mi. Procedural generation of game levels and maps: A review. In *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 050–055. IEEE, 2022.

[14] Linus Gisslén, Andy Eakins, Camilo Gordillo, Joakim Bergdahl, and Konrad Tollmar. Adversarial reinforcement learning for procedural content generation. In *2021 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2021.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[16] Daniele Gravina, Ahmed Khalifa, Antonios Liapis, Julian Togelius, and Georgios N Yannakakis. Procedural content generation through quality diversity. In *2019 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2019.

[17] Djordje Grbic, Rasmus Berg Palm, Elias Najarro, Claire Glanois, and Sebastian Risi. Evocraft: A new challenge for open-endedness. In *Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 24*, pages 325–340. Springer, 2021.

[18] Michael Cerny Green, Luvneesh Mugrai, Ahmed Khalifa, and Julian Togelius. Mario level generation from mechanics using scene stitching. In *2020 IEEE Conference on Games (CoG)*, pages 49–56. IEEE, 2020.

[19] Maxim Gumin. Wave Function Collapse Algorithm, 9 2016.

[20] Xiang Guo, Guanying Chen, Yuchao Dai, Xiaoqing Ye, Jiadai Sun, Xiao Tan, and Errui Ding. Neural deformable voxel grid for fast optimization of dynamic view synthesis. In *Proceedings of the Asian Conference on Computer Vision*, pages 3757–3775, 2022.

[21] Matthew Guzdial, Duri Long, Christopher Cassion, and Abhishek Das. Visual procedural content generation with an artificial abstract artist. In *Proceedings of ICCC computational creativity and games workshop*, 2017.

[22] Matthew Guzdial, Sam Snodgrass, and Adam J Summerville. Constraint-based pcgml approaches. In *Procedural Content Generation via Machine Learning: An Overview*, pages 51–66. Springer, 2022.

[23] Matthew Guzdial, Sam Snodgrass, and Adam J Summerville. Pcgml process overview. In *Procedural Content Generation via Machine Learning: An Overview*, pages 35–49. Springer, 2022.

[24] Matthew Guzdial, Sam Snodgrass, and Adam J Summerville. *Procedural Content Generation Via Machine Learning: An Overview*. Springer, 2022.

[25] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. Gancraft: Unsupervised 3d neural rendering of minecraft worlds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14072–14082, 2021.

[26] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1–22, 2013.

[27] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparametrization with gumble-softmax. In *International Conference on Learning Representations (ICLR 2017)*. OpenReview. net, 2017.

[28] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR, 2021.

[29] Zehua Jiang, Sam Earle, Michael Green, and Julian Togelius. Learning controllable 3d level generators. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, pages 1–9, 2022.

[30] Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle tower: A generalization challenge in vision, control, and planning. *arXiv preprint arXiv:1902.01378*, 2019.

[31] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. In *NeurIPS Workshop on Deep Reinforcement Learning*, 2018.

[32] Isaac Karth and Adam M Smith. Addressing the fundamental tension of pcgml with discriminative learning. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, pages 1–9, 2019.

[33] Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. Pcgrl: Procedural content generation via reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 95–101, 2020.

[34] Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

[35] Han-Hung Lee and Angel X Chang. Understanding pure clip guidance for voxel grid nerf models. *arXiv preprint arXiv:2209.15172*, 2022.

[36] Vivian Lee, Nathan Partlan, and Seth Cooper. Precomputing player movement in platformers for level generation with reachability constraints. In *AIIDE Workshops*, 2020.

[37] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):213–228, 2012.

[38] Antonios Liapis, Georgios Yannakakis, and Julian Togelius. Designer modeling for personalized game content creation tools. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 9, pages 11–16, 2013.

[39] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[40] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N Yannakakis, and Julian Togelius. Deep learning for procedural content generation. *Neural Computing and Applications*, 33(1):19–37, 2021.

[41] Cyprezz LLC. Planet minecraft community: Creative fansite for everything minecraft! https://www.planetminecraft.com/.

[42] Christian E López, James Cunningham, Omar Ashour, and Conrad S Tucker. Deep reinforcement learning for procedural content generation of 3d virtual environments. *Journal of Computing and Information Science in Engineering*, 20(5), 2020.

[43] Alejandro Medina, Melanie Richey, Mark Mueller, and Jacob Schrum. Evolving flying machines in minecraft using quality diversity. *arXiv preprint arXiv:2302.00782*, 2023.

[44] Timothy Merino, M Charity, and Julian Togelius. Interactive latent variable evolution for the generation of minecraft structures. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, pages 1–8, 2023.

[45] B Mildenhall, PP Srinivasan, M Tancik, JT Barron, R Ramamoorthi, and R Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, 2020.

[46] Nasir Mohammad Khalid, Tianhao Xie, Eugene Belilovsky, and Tiberiu Popa. Clip-mesh: Generating textured meshes from text using pretrained image-text models. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–8, 2022.

[47] Justin Mott, Saujas Nandi, and Luke Zeller. Controllable and coherent level generation: A two-pronged approach. In *Experimental AI in games workshop*, 2019.

[48] Rohit Nair. *Using Raymarched shaders as environments in 3D video games*. Drexel University, 2020.

[49] Mark J Nelson, Julian Togelius, Cameron Browne, and Michael Cook. Rules and mechanics. *Procedural Content Generation in Games*, pages 99–121, 2016.

[50] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pages 17473–17498. PMLR, 2022.

[51] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

[52] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[53] Sebastian Risi and Julian Togelius. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, 2(8):428–436, 2020.

[54] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, 6 2022.

[55] Christoph Salge, Michael Cerny Green, Rodgrigo Canaan, and Julian Togelius. Generative design in minecraft (gdmc) settlement generation competition. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, pages 1–10, 2018.

[56] Christoph Salge, Claus Aranha, Adrian Brightmoore, Sean Butler, Rodrigo De Moura Canaan, Michael Cook, Michael Green, Hagen Fischer, Christian Guckelsberger, Jupiter Hadley, et al. Impressions of the gdmc ai settlement generation challenge in minecraft. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, pages 1–16, 2022.

[57] Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. *arXiv preprint arXiv:2109.13202*, 2021.

[58] Anurag Sarkar and Seth Cooper. Sequential segment-based level generation and blending using variational autoencoders. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*, pages 1–9, 2020.

[59] Anurag Sarkar, Zhihan Yang, and Seth Cooper. Conditional level generation and game blending. *arXiv preprint arXiv:2010.07735*, 2020.

[60] Noor Shaker, Georgios Yannakakis, and Julian Togelius. Towards automatic personalized content generation for platform games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 6, pages 63–68, 2010.

[61] Noor Shaker, Julian Togelius, and Mark J Nelson. Procedural content generation in games. 2016.

[62] Noor Shaker, Julian Togelius, Mark J Nelson, Antonios Liapis, Gillian Smith, and Noor Shaker. Mixed-initiative content creation. *Procedural content generation in games*, pages 195–214, 2016.

[63] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. Make-a-video: Text-to-video generation without text-video data. *arXiv preprint arXiv:2209.14792*, 2022.

[64] Matthew Siper, Ahmed Khalifa, and Julian Togelius. Path of destruction: Learning an iterative level generator using a small dataset. *arXiv preprint arXiv:2202.10184*, 2022.

[65] Ole Edvin Skjeltorp. 3d neural cellular automata-simulating morphogenesis: Shape, color and behavior of three-dimensional structures. Master's thesis, 2022.

[66] Adam M Smith and Michael Mateas. Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):187–200, 2011.

[67] Shyam Sudhakaran, Djordje Grbic, Siyan Li, Adam Katona, Elias Najarro, Claire Glanois, and Sebastian Risi. Growing 3d artefacts and functional machines with neural cellular automata. *arXiv preprint arXiv:2103.08737*, 2021.

[68] Shyam Sudhakaran, Miguel González-Duque, Claire Glanois, Matthias Freiberger, Elias Najarro, and Sebastian Risi. Mariogpt: Open-ended text2level generation through large language models, 2023.

[69] Adam Summerville and Michael Mateas. Sampling hyrule: Multi-technique probabilistic level generation for action role playing games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 11, pages 63–67, 2015.

[70] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning (pcgml). *IEEE Transactions on Games*, 10(3):257–270, 2018.

[71] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022.

[72] Jiaxiang Tang. Stable-dreamfusion: Text-to-3d with stable-diffusion, 2022.

[73] Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

[74] Adaptive Agent Team, Jakob Bauer, Kate Baumli, Satinder Baveja, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, et al. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608*, 2023.

[75] Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius. Level generation through large language models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, pages 1–8, 2023.

[76] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.

[77] Ruben Rodriguez Torrado, Ahmed Khalifa, Michael Cerny Green, Niels Justesen, Sebastian Risi, and Julian Togelius. Bootstrapping conditional gans for video game level generation. In *2020 IEEE Conference on Games (CoG)*, pages 41–48. IEEE, 2020.

[78] Peihao Wang, Xuxi Chen, Tianlong Chen, Subhashini Venugopalan, Zhangyang Wang, et al. Is attention all nerf needs? *arXiv preprint arXiv:2207.13298*, 2022.

[79] Peng Wang, Yuan Liu, Zhaoxi Chen, Lingjie Liu, Ziwei Liu, Taku Komura, Christian Theobalt, and Wenping Wang. F^{2}-nerf: Fast neural radiance field training with free camera trajectories. *arXiv preprint arXiv:2303.15951*, 2023.

[80] Ryan Watkins. *Procedural content generation for unity game development*. Packt Publishing Ltd, 2016.

[81] Georgios N Yannakakis and Julian Togelius. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, 2(3):147–161, 2011.

[82] Cristopher Yates. *The use of Poisson Disc Distribution and A* Pathfinding for Procedural Content Generation in Minecraft*. PhD thesis, Ph. D. Dissertation. Memorial University, 2021.

[83] Hejia Zhang, Matthew Fontaine, Amy Hoover, Julian Togelius, Bistra Dilkina, and Stefanos Nikolaidis. Video game level repair via mixed integer linear programming. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 151–158, 2020.

[84] Alexander Zook and Mark O Riedl. Generating and adapting game mechanics. In *Proceedings of the 2014 Foundations of Digital Games Workshop on Procedural Content Generation in Games*, 2014.

A Minecraft Textures

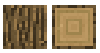
block	texture	block	texture	block	texture
log oak		stone		dirt	
brick		clay		snow	
glazed terracotta light blue		glazed terracotta yellow		redstone block	
gold block		iron block		diamond block	
emerald block		cobblestone		slime	

Table 4: In-game blocks and textures used by DreamCraft

B Quantization Schemes

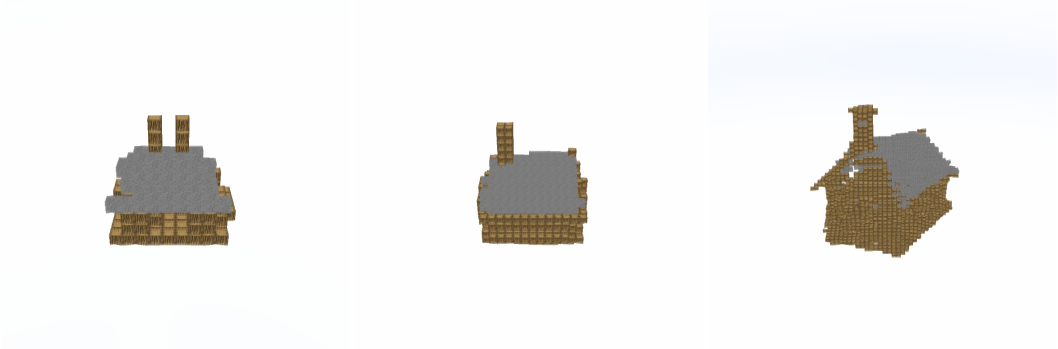
block type	block density	CLIP ViT-B/16 RGB	CLIP ViT-B/16 depth	CLIP ViT-B/32 RGB	CLIP ViT-B/32 depth
anneal	anneal	12.68	4.31	11.37	3.01
	hard	5.36	0.65	6.80	0.92
	soft	22.88	8.89	22.88	8.24
hard	anneal	12.68	5.88	14.12	4.71
	hard	4.05	2.22	4.97	1.18
	soft	19.61	**12.03**	21.83	**11.11**
soft	anneal	17.65	4.84	15.69	4.44
	hard	9.80	1.05	7.32	0.92
	soft	**24.31**	8.76	**24.97**	7.32

Table 5: Fidelity of generated structures given different quantization schemes for block density and type. COCO dataset, neural renders. A combination of fully discrete block types and "soft" block density leads to best performance. R-precision averaged over 5 poses for each experiment.

C Block Grid Resolution

We experiment with the resolution of the block grid, learning a $N \times N \times N$-block representation of text prompts with $N \in \{10, 20, \ldots, 100\}$.

In Table 6, we see that increasing block grid resolution leads to an increase in R-precision. We note that the more blocks in the grid, the closer each block comes to being represented by only a single pixel in each 2D render of the 3D block layout. In other words, we can expect the output of these higher-resolution quantized NeRFs to approximate that of their unconstrained counterparts with increasing accuracy. By analogy with visual art, we can say that the model uses blocks in an increasingly pointillistic fashion.



$N = 30$ $N = 50$ $N = 90$

Figure 7: *"medium medieval home"* with block grids of various widths.

block grid	CLIP ViT-B/16 RGB	CLIP ViT-B/16 depth	CLIP ViT-B/32 RGB	CLIP ViT-B/32 depth
10	3.87	4.53	3.07	2.67
20	5.47	2.40	6.00	2.93
30	6.53	5.60	9.33	3.47
40	6.53	6.13	8.67	6.93
50	6.40	6.93	6.27	5.07
60	7.33	6.80	9.33	4.40
70	7.87	8.53	11.47	6.80
80	11.73	8.27	12.27	6.67
90	10.40	8.67	10.93	**9.07**
100	**10.80**	**9.07**	**13.33**	8.67

Table 6: Fidelity of generated structures given block grids of varying resolutions. Planet Minecraft dataset. Neural renders. Performance increases with the resolution of the block grid.

D DreamCraft Generations

"A pile of crab is seasoned and well cooked"

Unconstrained NeRF | $N = 100$ | $N = 100$, in-game

"A pizza and fork on a tray on the table"

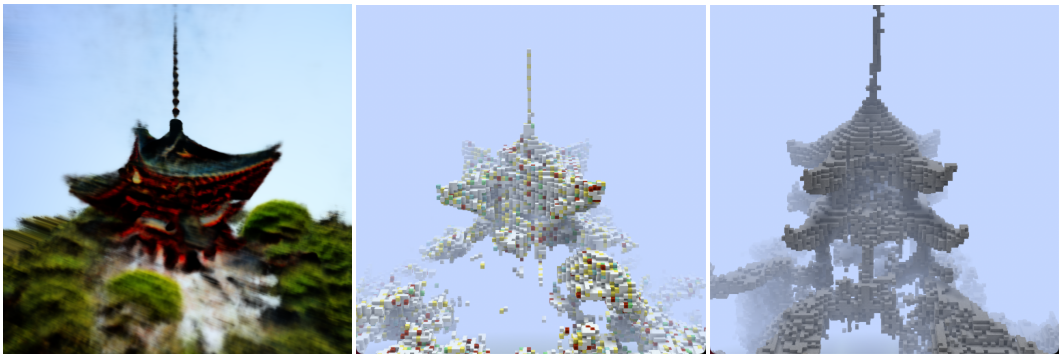
Figure 8: DreamCraft output given COCO dataset captions, viewed in-game (right), by neural rendering (middle), and standard DreamFusion output (left) given the same prompts.

E Limitations

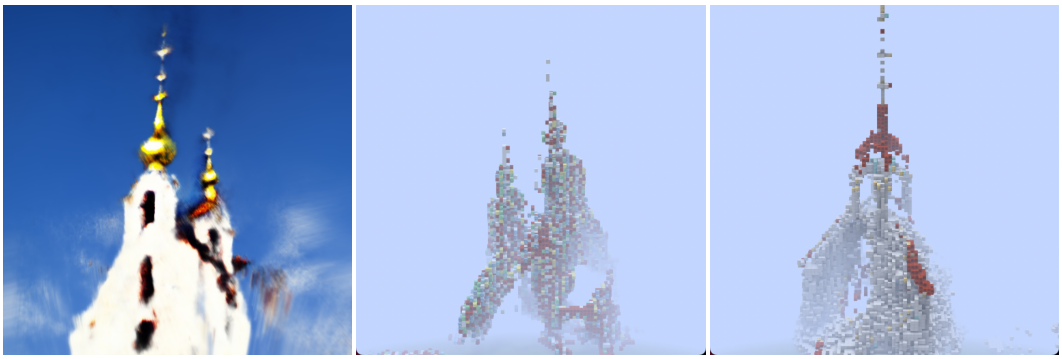
In some cases, the model uses negative space to represent an object, modulating the background texture to a particular color, then occluding parts of it with foreground blocks/density, to give it an apparent shape. This undesirable swapping of roles between foreground/background models may be more likely to occur in the quantized NeRF: whereas certain colors or textures may be difficult or impossible to replicate using the provided blocks, the background MLP remains unconstrained. To mitigate this, future work could investigate constraining the background MLP to only use 2D projections of "distant" game assets.

Another potential issue is the lack of semantic grounding with respect to block types. For example the model may just as well satisfy the prompt "large medieval ship" by using a combination of dirt and redstone, as with actual wooden logs or planks, so long as these give the *appearance* of wood. Our preliminary work on functional constraints suggests that this particular problem can be addressed by setting per-block-type targets (e.g. requiring 0% dirt and 50% wood blocks), but a more general approach might lie in "demonstrating" what each block type should be used to represent by adding this information in the prompt.

Whereas traditional NeRFs can model lighting and shadows, this is not the case in DreamCraft, where the color at each point in 3D space is derived directly from a voxel grid corresponding to the in-game appearance of a Minecraft block. When structures are rendered inside the neural engine, they thus appear "flat" in contrast to the kind of shadow and lighting effects that appear in the Minecraft game engine. Ideally, we could train an auxiliary model to mimic the effects of in-game lighting, for example by training it on paired datapoints of 3D block grids, and their appearance in-game at various angles. This learned renderer could replace the differentiable raycasting component of the NeRF pipeline (as in [78]), further sparing us from having to re-implement the rendering of irregular game objects such as plants and glass.



(a) *"a japanese temple"*, from the Planet Minecraft dataset.



unconstrained DreamFusion | DreamFusion with post hoc quantization using nearest neighbor mapping | DreamCraft

(b) *"church of the annunciation of the blessed virgin mary"*, from the Planet Minecraft dataset.

Figure 9: DreamCraft directly optimizes a representation using Minecraft blocks, leading to a more faithful reconstruction of the text prompt than results from post-processing the output of an unconstrained DreamFusion model.

DreamCraft is currently too slow to be feasibly used in an online player-environment generator loop, taking a few hours to generate a single structure. Future versions could benefit from recent and future speed improvements in NeRFs [79, 20, 71, 83]. Alternatively, it could be leveraged to generate a training set for a conditional, guidance-free generative models of game worlds.

F Planet Minecraft Dataset

To test *DreamCraft*'s ability to generate environments specific to the domain for which it was designed, we source text prompts from Planet Minecraft, a fan-operated site where users can upload and share custom content. We consider a subset of assets uploaded to the "Maps" category in 2016 (the year in which the most such assets were uploaded), and select the top 150 maps of this subset as measued by the number of user downloads. The prompts correspond to the names of these assets. We do not collect the assets themselves or any further data from the site.

- paris eiffel tower
- la valle dor by mrbatou download cinematic
- reims cathedral
- summit creative house
- mexican hacjenda
- coruscant senate building
- from my house to yours merry christmas

- the ziggurat
- polaris skyscraper 25
- the craftsmans abode pmc solo contest 4
- rustic fantasy house timelapse download
- skyscraper 31 ias
- chateau de silveberg
- fuminsh the city that never sleeps
- ontario tower
- dirt modern house
- farin rocks
- elven tower of the wise
- distorsion chunk challenge
- tours thiers nancy france skyscraper 3 ias
- luxury beach house
- space lighthouse
- central place modern office complex
- minecraft is a small world
- tiger ii 101 scale
- shurwyth snowlands download weareconquest
- bridges
- jurassic world v2 for jurassicraft 20 minecraft dinosaurs jurassic park isla nublar
- brynwalda survival map
- icarly set and nickelodeon studio
- battle of hoth map echo base star wars hoth map
- dream
- the little castle nebelburg
- steampunk island
- land of azorth
- jaws ride and amity village
- avatar base
- quartz tower 1
- small modern house 3 full interior
- minecraft disneyland 1965
- 2012 skyrim
- greenfield project neoclassical house
- 2012 sea dogs village
- eternal haven
- patronis
- japanese temple
- star labs the flash cw
- hub spawn
- church of the annunciation of the blessed virgin mary inowrocaw
- space needle accurate
- the dark city hokkaido

- a watch tower inspired by the game firewatch
- undertale
- white snowy castle
- avengers tower
- fantme villa modern house 2
- abandoned wild west
- greenfield building vista creek elementary school
- fantasy bundle level 25 special
- a modern house 1
- server spawn by infro_
- fantasy inspired village danjgames
- classic american farm
- the builders shrine chunk challange
- der eisendrache
- modern house by real architect
- ahzvels hq download re upload
- download a medieval detached farm showcase
- wg tower vice city
- calypso a modern villa
- tf2 egypt
- minigames map atlantis
- large medieval ship
- small cabin
- sustainable city
- large medieval ship
- sequoia valley 30
- kraehenfels survival version
- small hospital
- panem mc 1st quarter quell arena download
- a desperate and lonely wizards tower pmc chunk challenge entry lore
- paper mariocolour splash port prisma
- skyscraper planus
- hollywood residence
- a nordic mountain village
- old wizards tree mansion series 2 build 1read description
- hidden in the sand
- five nights at candys 2 roleplay map
- castle ardor
- epic server spawn
- medium medieval home
- medeival windmill 1102
- grand stadium pixelmon
- prison mine 2 with download
- babylon gardens

- the new world trade center 11
- 30days day 28 orcish butchers slaughter house
- old west home
- ark labs outpost 26
- icebornminigames map
- grand university
- medieval mountain castle
- minecraft maps
- victorian manor
- 30days day 8 dwarven entrance
- spherical greenhouse 18 19 110
- huge minecraft server spawn airidale
- big cottage
- greenfield typical victorian
- old fortress
- modern condoapartment
- small castle
- ss tropic a custom by prestogo
- two story old west shop
- citadel hill fort george
- batman arkham asylum
- forest cottage
- mountain temple
- the conjuring
- northwich
- the amazing word of gumball the wattersons house 18
- five nights at freddys minecraft map 19 and above
- fnaf roleplay map
- middle eastern farm
- modern house build sorry about no music in the video
- a medieval farm
- woodland mansion
- survival map jairus isle
- mario kartgba bowsers castle 2
- the scandinavian townhouse
- mesa fort
- three cool wood structures
- brenttwood estates
- toy shop 192
- dota 2
- server shop
- store fletchers retreat
- custom realistic terrain
- the piggy sphinx

- shubbles castle building contest
- mountain housecastle 1
- roman outpost
- fallout 4 red rocket
- fantasy house
- savanna village in the sky cinematic download
- kahuai city
- minecraft lets build timelapse fantasy update 12 over hanging house
- kent regional airport
- medieval house
- redstone bunker 13 redstone creations version 1