Procedural Terrain Generation via GFlowNets:
Sampling Control Points for Blender Noise Functions

by

JayCe Leonard

A thesis accepted and approved in partial fulfillment of the
requirements for the degree of
Master of Science
in Computer Science

Thesis Committee:
Yu Wang, Chair
Jieyang Chen, Member

University of Oregon

Fall 2025

ABSTRACT

JayCe Leonard

Master of Science in Computer Science

Title: Procedural Terrain Generation via GFlowNets: Sampling Control Points for Blender Noise Functions

We present a novel application of Generative Flow Networks (GFlowNets) to procedural terrain generation in Blender. Rather than learning generative models from data, we train GFlowNets to explore parameters within existing procedural frameworks, specifically sampling Perlin noise configurations and color ramp control points that map noise values to terrain elevations. We compare Trajectory Balance (TB) and Sibling-Augmented GFlowNets (SA-GFN) to evaluate how efficiently different training objectives discover valid parameter configurations compared to random sampling, demonstrating their effectiveness at navigating high-dimensional procedural parameter spaces while maintaining solution diversity.

SA-GFN uses a dual-network architecture with Random Network Distillation for intrinsic motivation to address exploration challenges in sparse reward settings. We assess whether this decoupled exploration strategy improves mode discovery. Unlike recent work using learned latent representations, our method operates directly on interpretable procedural parameters, enabling artists to inspect and modify generated solutions without requiring training data collection or black-box generative models.

Contents

List of Figures

1 INTRODUCTION

Figure 1: A procedurally generated terrain mesh in Blender's 3D viewport. This work trains GFlowNets to sample the procedural parameters that produce such heightmaps, enabling diverse terrain generation without learned generative models.

Procedural modeling constructs complex structures by chaining together predefined operations. Each operation applies a specific transformation, and the sequence of operations defines the final output. This approach offers a key advantage over black-box generative methods: every output corresponds to explicit, inspectable parameters that artists can understand, modify, and reuse. The technique is widely used in computer graphics, game development, and engineering, where this interpretability, combined with flexibility and consistent results, is essential.

A common problem arises when designers have a working procedural framework but must adjust many control points to achieve a desired result. The procedural system might offer dozens of parameters controlling noise characteristics, elevation mappings, and detail distributions. Finding configurations that produce outputs with specific properties requires either extensive manual exploration or random search. Yet abandoning procedural workflows for data-driven generative models sacrifices the interpretability that makes procedural systems valuable in the first place.

Generative Flow Networks (GFlowNets) offer a middle path. Rather than learning to generate outputs directly, GFlowNets learn to sample configurations proportionally to a reward function, discovering diverse solutions rather than collapsing to a single optimum. GFlowNets and

procedural workflows share a structural similarity: both operate on directed acyclic graphs, with states representing partial constructions and edges representing actions. This correspondence suggests that GFlowNets can automate parameter search while preserving the interpretable structure of procedural systems.

This paper applies GFlowNets to procedural terrain generation in Blender. We train GFlowNets to navigate the parameter space of an existing procedural system, sampling Perlin noise configurations and color ramp control points that map noise values to terrain elevations. Every generated terrain corresponds to explicit, adjustable procedural parameters, not learned latent codes.

Contributions. Our work pursues three objectives:

1. We provide an implementation of GFlowNets operating through the Blender API, demonstrating that these methods can integrate with production content creation tools.

2. We produce a structured dataset of trajectories, heightmaps, and procedural configurations for other researchers to use.

3. We explore whether GFlowNets can viably sample from spaces of highly interpretable procedures, preserving the transparency that makes procedural workflows valuable while automating the search for configurations that satisfy specified objectives.

2 BACKGROUND

2.1 Generative Flow Networks

A directed graph $G = (S, A)$ consists of a finite set of states S and directed edges $A \subseteq S \times S$, where $s \to s'$ indicates a transition from state s to state s'. A trajectory $\tau = \{s_1, \ldots, s_n\}$ is a sequence of states connected by valid transitions, meaning each consecutive pair satisfies $s_t \to s_{t+1} \in A$.

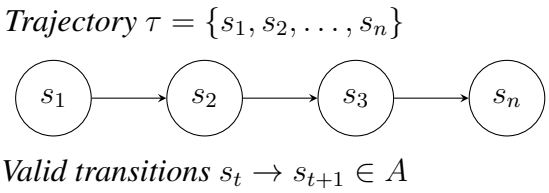

Figure 2: A trajectory τ as a sequence of connected states where $s_t \to s_{t+1} \in A$ for each transition.

Generative Flow Networks (GFlowNets)[1] learn policies that sample composite objects with probability proportional to a reward function. Where reinforcement learning seeks a single optimal policy that maximizes expected reward, GFlowNets sample diversely from the distribution $p(x) \propto R(x)$.

The framework operates on a directed acyclic graph $\mathcal{G} = (\mathcal{S}, \mathcal{A})$ with a unique source node s_0 and terminal states representing completed objects. The learnable components are a forward policy $P_F(s'|s)$ that selects actions and a backward policy $P_B(s|s')$ that reasons about how states were reached. Training enforces flow consistency: the probability mass flowing into any state must equal the mass flowing out, with terminal states receiving mass proportional to their rewards.

The Trajectory Balance (TB) objective[2] enforces this consistency across complete trajectories:

$$Z_\theta \prod_{t=0}^{n-1} P_F(s_{t+1}|s_t;\theta) = R(s_n) \prod_{t=1}^{n} P_B(s_{t-1}|s_t;\theta) \tag{1}$$

where Z_θ is a learned partition function. The training loss minimizes the squared log-ratio:

$$\mathcal{L}_{TB} = \left(\log Z_\theta + \sum_{t=0}^{n-1} \log P_F(s_{t+1}|s_t) - \log R(s_n) - \sum_{t=1}^{n} \log P_B(s_{t-1}|s_t) \right)^2 \tag{2}$$

2.2 Sibling Augmented GFlowNets

GFlowNets face exploration challenges in sparse reward settings where high-reward regions are separated by large low-reward spaces. The forward policy may never discover distant

[1]Emmanuel Bengio et al., "Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation," arXiv preprint arXiv:2106.04399 (2021).

[2]Nikolay Malkin et al., "Trajectory Balance: Improved Credit Assignment in GFlowNets," arXiv preprint arXiv:2201.13259 (2022).

modes through on-policy exploration alone.

Sibling Augmented Generative Flow Networks (SA-GFN)[3] address this through a dual-network architecture. The Behavior Network learns the true target distribution. The Sibling Network uses intrinsic motivation to explore beyond known rewards, sampling trajectories according to:

$$r_\tau^{SN} = \left((r^e)^{\beta_e^{SN}} + \left(\sum_t r_t^i \right)^{\beta_i} \right)^{\beta_{SN}} \tag{3}$$

where r^e is the true task reward, r_t^i are intrinsic rewards, and temperature parameters control the balance. Setting $\beta_e^{SN} = 0.25$ encourages the Sibling Network to explore beyond high-reward regions it has already discovered.

The intrinsic rewards come from Random Network Distillation (RND)[4], which measures novelty as the prediction error between a fixed random network $\bar{\phi}$ and a learned predictor ϕ:

$$r^i(s) = \|\bar{\phi}(s) - \phi(s)\|^2 \tag{4}$$

States encountered frequently become predictable and yield low intrinsic reward, while novel states produce high prediction errors that incentivize exploration.

The key insight enabling SA-GFN is that exploratory trajectories from the Sibling Network can be relabeled with true extrinsic rewards and used to train the Behavior Network. This leverages the off-policy learning capabilities of GFlowNets: the Behavior Network need not have generated a trajectory itself to learn from it. The result is a system where exploration and exploitation proceed in parallel, with the Sibling Network scouting new territory while the Behavior Network consolidates knowledge of the reward landscape.

[3] Moksh Madan et al., "Sibling-Augmented Generative Flow Networks," arXiv preprint (2025).

[4] Yuri Burda et al., "Exploration by Random Network Distillation," arXiv preprint arXiv:1810.12894 (2018).

3 RELATED WORK

GFlowNets were developed for molecular discovery,[5] where sampling diverse molecular structures proportional to predicted binding affinities offered advantages over optimization methods that collapse to single candidates. This application and related work in biological sequence design[6] operate directly in discrete combinatorial spaces without requiring learned representations.

Recent applications to content generation couple GFlowNets with learned latent spaces. Rainbow[7] trains GFlowNets to sample trajectories over latent graphs for conditional image generation. Zhang and Yang[8] use a two-stage pipeline: first training a conditional GAN on USGS elevation data, then training a GFlowNet to sample control points that condition the generator.

Our work avoids neural generative models entirely, applying GFlowNets directly to Blender's procedural framework where the mapping from parameters to outputs is deterministic. No training data is required because the procedural system provides generative capacity through mathematical functions. Each parameter's effect is transparent and adjustable. Recent theoretical work[9] suggests GFlowNets leverage structure in environment dynamics, and procedural frameworks provide this structure through explicit mathematical operations.

4 METHOD

4.1 Blender and Geometry Nodes

Blender is an open-source 3D modeling application widely used in film, game development, and visualization. Unlike proprietary tools, Blender provides a complete Python API that exposes nearly all functionality programmatically. This accessibility makes Blender suitable for research

[5]Bengio et al., "Flow Network based Generative Models."

[6]Lena Podina et al., "Catalyst GFlowNet for Electrocatalyst Design: A Hydrogen Evolution Reaction Case Study," arXiv preprint arXiv:2510.02142 (2024).

[7]Authors et al., "Discovering Latent Graphs with GFlowNets for Diverse Conditional Image Generation," arXiv preprint arXiv:2510.22107 (2024).

[8]Chong Zhang and Lizhi Yang, "Generating a Terrain-Robustness Benchmark for Legged Locomotion: A Prototype via Terrain Authoring and Active Learning," arXiv preprint arXiv:2208.07681 (2022).

[9]Lazar Atanackovic and Emmanuel Bengio, "Investigating Generalization Behaviours of Generative Flow Networks," Transactions on Machine Learning Research (2025).

applications where automated control over the modeling pipeline is required.

Blender's Geometry Nodes system enables procedural design through a visual interface where nodes represent operations and connections define data flow. Each node performs a specific function: generating primitives, transforming geometry, sampling textures, or combining data streams. The editor forms a directed acyclic graph (DAG), where information flows from input nodes through processing stages to output nodes without cycles. Artists construct complex procedural systems by wiring together simple operations, with each connection carrying geometry, numerical values, or other data types.

4.2 Terrain Generation Pipeline

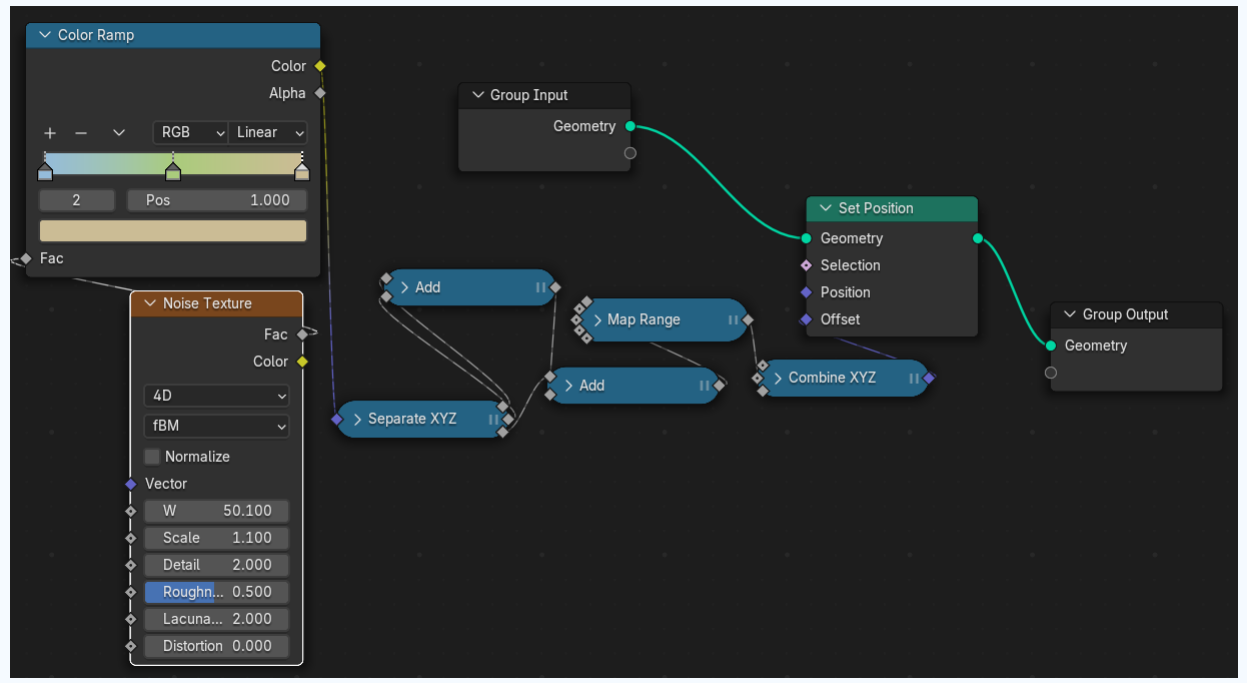

Figure 3: Blender Geometry Nodes graph defining the procedural terrain generation workflow. The Color Ramp node (left) maps noise values to elevation, with control points that determine the height distribution. The Noise Texture node generates 4D fBM noise with parameters W and Scale that control the spatial pattern. These nodes feed through mathematical operations to Set Position, displacing geometry vertices to create the final heightmap.

Figure 3 shows the complete node graph for terrain generation. The system operates through two components arranged in sequence. A Perlin noise texture generates the base elevation

field, producing values in $[0, 1]$ across a 2D grid. Two parameters control this texture: W selects a region of the infinite noise field, effectively choosing a random seed, and scale determines the spatial frequency of features. Lower scale values produce broad, rolling hills; higher values produce fine, detailed terrain.

A color ramp maps noise values to final heights through control points (p_i, v_i). Each control point defines a position $p_i \in [0, 1]$ along the input range and an output value $v_i \in [0, 1]$ determining the corresponding elevation. The ramp interpolates linearly between control points, transforming the uniform noise distribution into a shaped elevation profile. By placing control points strategically, an artist can create plateaus, cliffs, or gradual slopes from the same underlying noise pattern.

The composition of noise generation followed by value remapping produces the final heightmap:

$$H(x, y) = \mathcal{R}\big(\mathcal{N}(x, y; W, \text{scale})\big) \tag{5}$$

where $\mathcal{N}$ is the noise function and $\mathcal{R}$ is the color ramp transfer function defined by the control points.

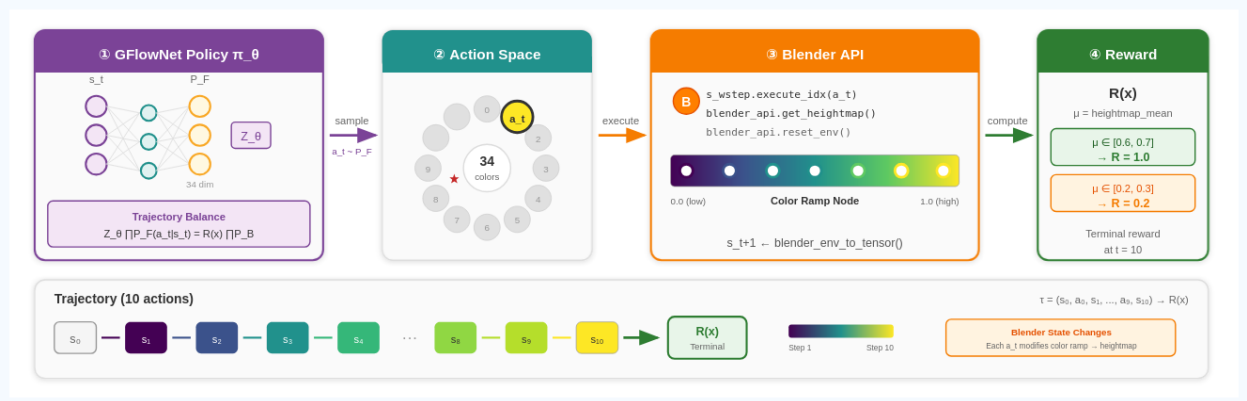

Figure 4: GFlowNet terrain generation pipeline. The policy π_θ samples actions $a_t \sim P_F(\cdot|s_t)$ from the action space. Each action modifies control points on Blender's color ramp node, updating the procedural heightmap. After 10 actions, the terminal reward $R(x)$ is computed from heightmap properties. The model is trained with Trajectory Balance: $Z_\theta \prod P_F(a_t|s_t) = R(x) \prod P_B$.

4.3 State and Action Space

The GFlowNet state captures the current procedural configuration:

$$s_t = (W_t, \text{scale}_t, \{(p_i, v_i)\}_i, t) \tag{6}$$

where t indexes the current step in the trajectory. The initial state s_0 contains default noise parameters and an empty color ramp.

Table 1: Action space definition.

Action	Type	Effect
0	step_w	Increment W parameter by 2.0
1	step_scale	Increment scale by 0.1
2–33	add_color	Add predefined control point to ramp
34	stop	Terminate trajectory

The action space consists of 35 discrete actions (Table 1). Actions a_0 and a_1 modify the noise parameters, shifting to different noise regions or adjusting spatial frequency. Actions a_2 through a_{33} each add a predefined control point (p_k, v_k) to the color ramp. This discretization converts the continuous space of possible control points into a finite action set that the GFlowNet can learn over. Once a color is added to the ramp, the corresponding action is masked to prevent duplicate control points. Action a_{34} terminates the trajectory. After 10 actions or upon selecting the termination action, the trajectory ends and the reward is computed from the resulting heightmap.

Figure 5 illustrates a complete trajectory. The left panel shows the action sequence as a directed graph, with colors indicating temporal order. The center panel displays the resulting heightmap. The right panel summarizes action statistics for the trajectory.

4.4 Reward Functions

We define two reward functions targeting different heightmap properties.

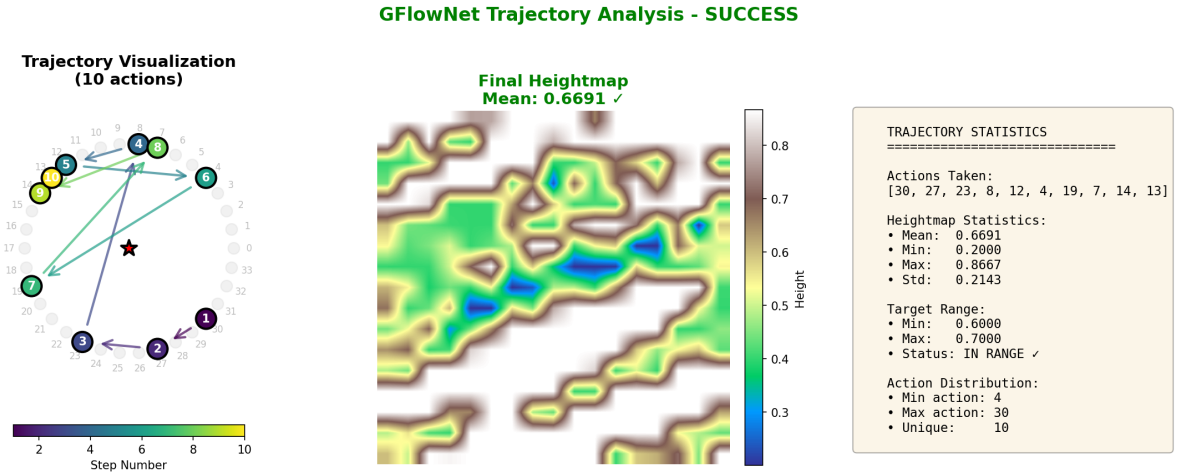

Figure 5: Successful trajectory achieving heightmap mean 0.6922 in range [0.6, 0.7]. **Left:** Action sequence as directed graph (purple=early, yellow=late, red star=terminal). **Center:** Final heightmap (blue=low ∼0.4, yellow=high ∼0.9). **Right:** Statistics showing 10 unique actions with action 32 most frequent.

Average Elevation. The first reward targets specific mean elevation values:

$$R_{\text{elev}}(\tau) = \begin{cases} 1.0 & \text{if } \bar{h} \in [0.6, 0.7] \text{ (high terrain)} \\ 0.2 & \text{if } \bar{h} \in [0.2, 0.3] \text{ (low terrain)} \\ 0.0 & \text{otherwise} \end{cases} \tag{7}$$

where $\bar{h} = \frac{1}{N}\sum_i^N h_i$ is the mean heightmap value normalized to $[0, 1]$. The two distinct target ranges test multi-modal sampling, requiring the model to discover and maintain coverage of both high and low terrain configurations. The different reward magnitudes (1.0 vs 0.2) test whether the model samples proportionally to reward as theory predicts.

Frequency Content. The second reward targets spatial frequency characteristics based on FFT analysis. The low frequency ratio measures the proportion of spectral energy in low frequencies (broad, smooth terrain features), while the high frequency ratio captures energy in high frequencies

(fine detail and texture):

$$R_{\text{freq}}(\tau) = \begin{cases} 1.5 & \text{if (low\_freq} > 0.95) \wedge (\text{high\_freq} < 0.05) \text{ (SMOOTH)} \\ 1.0 & \text{if (low\_freq} \in [0.75, 0.85]) \wedge (\text{high\_freq} \in [0.08, 0.12]) \text{ (DETAILED)} \\ 0.0 & \text{otherwise} \end{cases} \tag{8}$$

4.5 Training

Training follows the SA-GFN framework with trajectory balance loss. The system maintains two GFlowNet policies (Behavior and Sibling), an RND network for intrinsic rewards, and two replay buffers. Algorithm 4.5 presents the training procedure.

[h] SA-GFN Training with Blender Environment [1] Behavior Network π^{BN}, Sibling Network π^{SN}, RND networks $(\bar{\phi}, \phi)$ Replay buffers $\mathcal{B}_{BN}$, $\mathcal{B}_{SN}$; Blender API; trajectory length $T = 10$

each epoch $k = 1$ to $N/2$ Behavior trajectories $\tau \leftarrow$ SAMPLEBLENDER(π^{BN}, T) $\mathcal{B}_{BN} \leftarrow \mathcal{B}_{BN} \cup \{\tau\}$ $k = 1$ to $N/2$ Sibling trajectories with RND $\tau \leftarrow$ SAMPLEBLENDER$(\pi^{SN}, T, \text{rnd} = \phi)$ $\mathcal{B}_{SN} \leftarrow \mathcal{B}_{SN} \cup \{\tau\}$ Update ϕ on visited states: $\mathcal{L}_{RND} = \|\bar{\phi}(s) - \phi(s)\|^2$ Update π^{BN} on $\mathcal{B}_{BN} \cup \mathcal{B}_{SN}$ with extrinsic rewards (relabeling) Update π^{SN} on $\mathcal{B}_{SN}$ with combined rewards: $r^{SN} = ((r^e)^{0.25} + \sum r^i)$

SampleBlenderπ, T, rnd=None Blender.reset_env() $s_0 \leftarrow$ Blender.state_to_tensor() $\mathcal{U} \leftarrow \emptyset$ Used actions $t = 0$ to $T - 1$ $P_F \leftarrow \pi(s_t)$; mask actions in $\mathcal{U}$ Sample $a_t \sim \text{softmax}(P_F)$; $\mathcal{U} \leftarrow \mathcal{U} \cup \{a_t\}$ rnd $\neq$ None $r_t^i \leftarrow \|\bar{\phi}(s_t) - \phi(s_t)\|^2$ Blender.execute(a_t) $s_{t+1} \leftarrow$ Blender.state_to_tensor() $r^e \leftarrow R$(Blender.get_heightmap()) $\tau = (s_{0:T}, a_{0:T-1}, r^e, \{r_t^i\})$

Each trajectory involves direct interaction with Blender through a Python API. The RESET_ENV call clears the color ramp and resets noise parameters. The EXECUTE call modifies the procedural node graph—either adjusting noise parameters (a_0, a_1) or adding control points to the color ramp (a_2–a_{33}). After the trajectory completes, the heightmap is extracted and the reward computed.

Action masking prevents duplicate control points: once a color is added, its action is masked for the remainder of the trajectory. States are encoded as fixed-size tensors combining one-hot encodings for discrete W values, continuous scale values, and binary indicators for occupied color ramp positions.

The key mechanism enabling SA-GFN is trajectory relabeling: Sibling Network trajectories are collected using intrinsic rewards for exploration, but relabeled with true extrinsic rewards when training the Behavior Network. This allows the Behavior Network to learn from exploratory data without being influenced by the non-stationary intrinsic reward signal. Following the SA-GFN paper, we set $\beta_e^{SN} = 0.25$ to encourage exploration beyond discovered modes.

5 EXPERIMENTS

We evaluate GFlowNet terrain generation on the two reward functions defined in Section 4.4. All experiments use trajectory balance training with experience replay, comparing trained models against random sampling baselines.

5.1 Experimental Setup

The action space consists of 35 discrete actions, yielding a combinatorially large trajectory space. With 35 possible actions at each step, the number of unique trajectories of length n is 35^n. For trajectories of length 10, this produces $35^{10} \approx 2.76 \times 10^{15}$ possible action sequences. Even accounting for early termination via the stop action, the effective search space remains intractable for exhaustive enumeration.

We fix the trajectory length at 10 actions for all experiments. This choice balances expressiveness with computational tractability: shorter trajectories limit the complexity of achievable color ramp configurations, while longer trajectories exponentially increase training time and sample complexity.

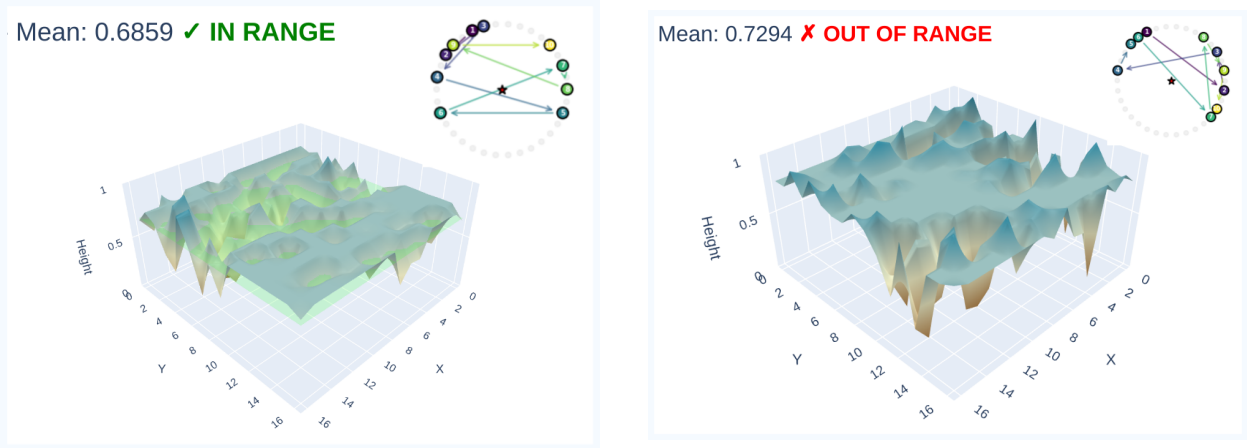

Figure 6: Example heightmaps with their action trajectories. Left: A successful sample with mean 0.6859 falling within the target range $[0.6, 0.7]$, receiving reward $R = 1.0$. Right: An unsuccessful sample with mean 0.7294 just outside the target range, receiving reward $R = 0$. The trajectory graphs (upper right of each) show the sequence of actions taken, with node colors indicating temporal order.

5.2 Average Elevation Results

Figure 6 illustrates the binary nature of the reward: small differences in mean elevation (0.6859 vs 0.7294) determine whether a trajectory receives reward or not.

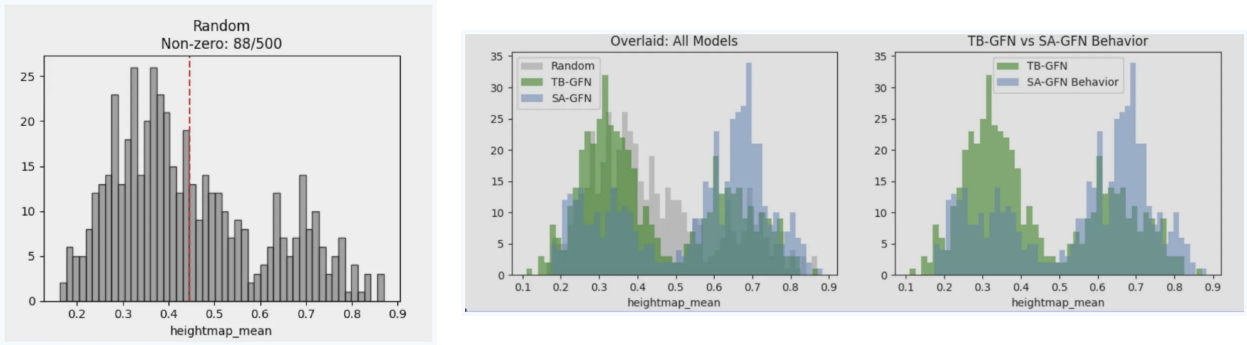

Figure 7: Left: Random sampling baseline over 500 samples, showing broad distribution with only 88 samples (17.6%) in target regions. Right: Overlaid distributions comparing all models (left panel) and TB-GFN vs SA-GFN Behavior (right panel). Trained models exhibit clear bimodal concentration in the target ranges $[0.2, 0.3]$ and $[0.6, 0.7]$.

Figure 7 compares the sampling distributions. Random sampling produces a broad spread across heightmap means with slight bias toward lower elevations. Both TB-GFN and SA-GFN learn to concentrate in the target regions, with SA-GFN Behavior showing stronger concentration

in the high-reward $[0.6, 0.7]$ range.

Table 2: Average elevation experiment results across 500 samples per model.

Model	Non-zero Rewards	Success Rate	vs Random
Random	88	17.6%	—
TB-GFN	104	20.8%	+18.2%
SA-GFN Behavior	138	27.6%	+56.8%
SA-GFN Sibling	89	17.8%	+1.1%
SA-GFN Behavior vs TB-GFN: +32.7%			

Table 2 summarizes the quantitative results. SA-GFN Behavior achieves the highest success rate at 27.6%, representing a 56.8% improvement over random sampling and a 32.7% improvement over TB-GFN. The SA-GFN Sibling network, which uses intrinsic motivation to prioritize exploration, performs comparably to random sampling in terms of reward acquisition. This is expected behavior: the Sibling network's role is to discover novel states rather than maximize reward, and its exploratory trajectories are relabeled to train the Behavior network.

5.3 Frequency Content Results

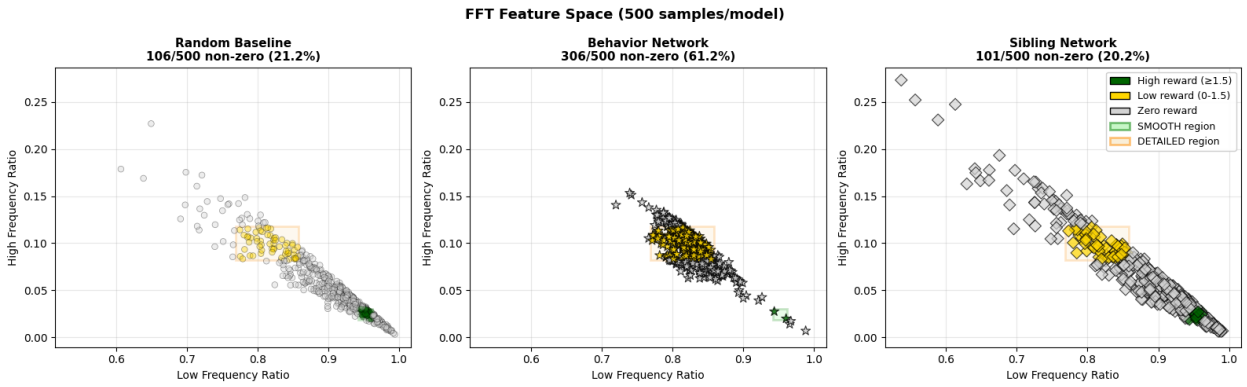

Figure 8: FFT feature space comparison across 500 samples per model. The SMOOTH region (green box, lower right) carries reward 1.5, while the DETAILED region (orange box) carries reward 1.0. Despite the higher reward for SMOOTH terrain, the Behavior Network exhibits mode collapse onto the DETAILED region.

Figure 8 and Table 3 reveal challenges in amortizing sampling over more complex multi-modal reward landscapes. Figures 9 and 10 illustrate the qualitative difference between target

Table 3: Frequency content experiment results across 500 samples per model.

Model	Non-zero	SMOOTH (R=1.5)	DETAILED (R=1.0)	Success Rate
Random	106	17	89	21.2%
Behavior Network	306	12	294	61.2%
Sibling Network	101	14	87	20.2%

Figure 9: Example terrain from the SMOOTH region (R=1.5), characterized by gentle undulations and low spectral centroid (SC $\approx$ 0.007). These configurations produce rolling landscapes with broad elevation changes and minimal fine detail.

regions: SMOOTH terrain exhibits gentle, rolling landscapes dominated by low-frequency content, while DETAILED terrain shows sharp peaks and fine texture from higher frequency components.

The Random baseline scatters broadly with 106/500 (21.2%) hitting target regions. The Behavior Network achieves 306/500 (61.2%) non-zero rewards but exhibits clear mode collapse: despite the SMOOTH region carrying higher reward (1.5 vs 1.0), the network concentrates almost entirely on the DETAILED region (294 samples) and underexplores the SMOOTH target (only 12 samples). This suggests that when target modes are spatially separated in feature space and require qualitatively different parameter configurations to reach, SA-GFN struggles to maintain proportional sampling across modes. The Sibling Network achieves 101/500 (20.2%) non-zero rewards with broader coverage of the feature space, but its exploratory trajectories fail to sufficiently guide the Behavior Network toward the higher-reward SMOOTH region.

6 CONCLUSION

Future directions span three areas. First, longer trajectory lengths would enable more complex terrain configurations, but naively increasing trajectory length exacerbates exploration

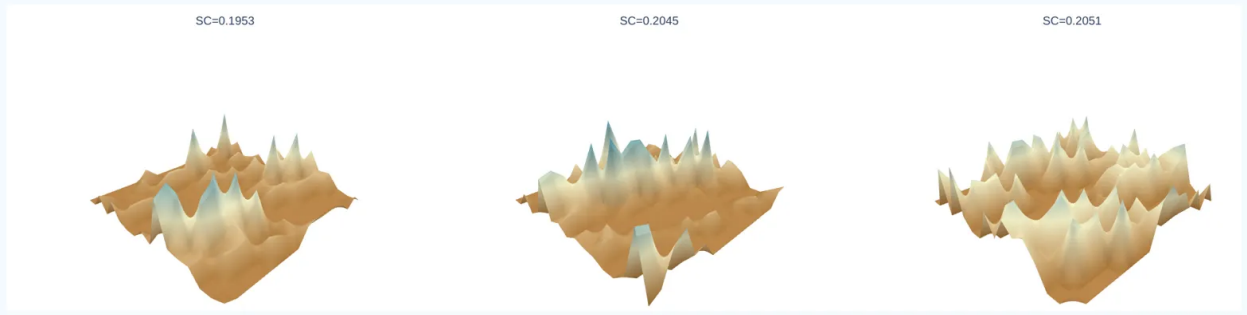

Figure 10: Example terrain from the DETAILED region (R=1.0), showing sharp peaks and fine texture with higher spectral centroid (SC $\approx$ 0.20). Despite carrying lower reward than the SMOOTH region, the Behavior Network collapses onto this mode.

challenges. Curriculum strategies that gradually extend trajectories during training—starting with short sequences where reward signal is dense, then lengthening as the policy improves—may help the model learn compositional structure before confronting the full search space.

Second, the mode collapse observed in the frequency content experiment suggests that SA-GFN's exploration mechanism, while effective for discovering modes, may not sufficiently encourage the Behavior Network to maintain coverage across structurally dissimilar targets. Alternative approaches might include explicit diversity bonuses in the reward function, ensemble methods that maintain separate specialists for different modes, or hierarchical policies that first select a target mode then specialize within it.

Third, the current action space covers only a fraction of Blender's procedural capabilities. Extended action spaces could incorporate noise octaves (controlling the fractal complexity of terrain), turbulence parameters, erosion operators that simulate hydraulic and thermal weathering, or vegetation distribution rules. Each extension multiplies the dimensionality of the search space but also the expressiveness of achievable terrains. The interpretability advantage of procedural methods scales with this expressiveness: a terrain defined by 50 meaningful parameters remains more transparent than one decoded from a 512-dimensional latent vector.

Beyond these immediate extensions, the broader question is whether GFlowNets can scale to the full complexity of professional procedural workflows, which may involve hundreds of interconnected nodes and thousands of parameters. The success on simpler configurations

demonstrated here provides initial evidence, but production-scale procedural systems will require advances in both GFlowNet training efficiency and reward specification—the latter being its own design challenge when artists may not know precisely what properties they want until they see examples. Code and data for this work are available online.[10]

[10] https://github.com/JayCeLeonardUO/bpygfn

REFERENCES CITED

Atanackovic, Lazar, and Emmanuel Bengio. "Investigating Generalization Behaviours of Generative Flow Networks." *Transactions on Machine Learning Research*, 2025.

Bengio, Emmanuel, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. "Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation." arXiv preprint arXiv:2106.04399, 2021.

Burda, Yuri, Harrison Edwards, Amos Storkey, and Oleg Klimov. "Exploration by Random Network Distillation." arXiv preprint arXiv:1810.12894, 2018.

Madan, Moksh, et al. "Sibling-Augmented Generative Flow Networks." arXiv preprint, 2025.

Malkin, Nikolay, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. "Trajectory Balance: Improved Credit Assignment in GFlowNets." arXiv preprint arXiv:2201.13259, 2022.

Podina, Lena, et al. "Catalyst GFlowNet for Electrocatalyst Design: A Hydrogen Evolution Reaction Case Study." arXiv preprint arXiv:2510.02142, 2024.

Rainbow Authors et al. "Discovering Latent Graphs with GFlowNets for Diverse Conditional Image Generation." arXiv preprint arXiv:2510.22107, 2024.

Zhang, Chong, and Lizhi Yang. "Generating a Terrain-Robustness Benchmark for Legged Locomotion: A Prototype via Terrain Authoring and Active Learning." arXiv preprint arXiv:2208.07681, 2022.