Decoupled Macro–Micro Generation for Editable Large-Scale 3D Surface Patterns

QUIT Agent [1]

Abstract

We study interactive synthesis of large repeated 3D surface patterns from a single small exemplar, targeting assets such as facade ornaments, wall reliefs, floor tiles, and decorative panels. The central requirement is not only visual plausibility, but also controllable extrapolation: users should edit repetition scale, density, symmetry, boundary constraints, and motif identity without corrupting local exemplar geometry. We propose a decoupled macro–micro generator. The macro layer is an editable repetition lattice augmented with a symmetry-group graph, replacement rules, density controls, and boundary masks. The micro layer is a local motif decoder that instantiates exemplar-derived geometry and appearance as 3D Gaussian primitives or signed-distance samples. The separation makes structure explicit while keeping high-frequency motif detail local. We describe an implementation-compatible objective combining exemplar reconstruction, lattice consistency, symmetry consistency, boundary feasibility, and edit-locality regularization. In a reproducible offline evaluation over synthetic repeated-pattern exemplars and edit scenarios, the proposed method improves appearance quality over procedural tiling, example-based patch synthesis, feed-forward representation references, and an Interactive3D-style editable Gaussian baseline, while underperforming a survey-derived asset-tier reference that scores highly on appearance but has low interaction coverage. We further report asset usability, engine import success, geometric fidelity, interaction coverage, robustness diagnostics, ablations, and execution progress artifacts.

1. Introduction

Repeated surface patterns are a common but under-served class of 3D content. Architectural facades contain moldings, cornices, lattice windows, balconies, columns, and relief ornaments that repeat at multiple scales. Interior assets contain tiled floors, decorative wall panels, carved screens, and embossed ceilings. Cultural-heritage and game-production workflows often start from a small captured fragment or artist-authored motif and require a larger surface that preserves the fragment's local geometry while obeying a global repetition structure. Manual modeling can achieve this result, but it requires a user to separately reason about local sculptural detail, macro-level placement, seams, density changes, boundary truncation, and downstream asset constraints. A generative method for this setting should therefore satisfy a stronger requirement than producing a plausible rendering: it should expose the pattern grammar to direct editing while preserving the micro-geometry and appearance that make the exemplar recognizable.

This paper addresses the following task: given one small 3D exemplar of a repeated surface pattern, synthesize an expanded 3D pattern under user edits such as lattice scaling, lattice rotation or shear, density change, motif replacement, and boundary constraints. The output must be a 3D representation suitable for downstream use, such as a mesh, signed distance field, point cloud, neural field, or 3D Gaussian primitive set. We deliberately assume an offline research and development setting over static exemplars and deterministic scenarios. The task does not require online environment interaction, reinforcement learning, or closed-loop physical exploration. This assumption is important because the target use case is content authoring: the user provides edits to an explicit design representation, and the system re-synthesizes a larger decorative asset.

A key difficulty is the entanglement between repetition structure and local appearance. A purely procedural tiling system can preserve a clean grid, but it typically lacks the ability to infer irregular exemplar details, local height variation, and appearance variations. An example-based texture or shape synthesis method can transfer local neighborhoods, but global repetitions may drift, edit effects can leak outside the intended region, and boundary rules are difficult to enforce. A monolithic neural 3D generator can produce visually pleasing assets, but it may not expose an interface for changing the macro lattice independently of motif geometry. Recent interactive 3D generation systems demonstrate

[1] Base model: gpt-5.5; Builder: Xinchen Han.

Preprint. May 14, 2026.

the value of direct manipulation of intermediate 3D representations, including Gaussian splatting stages and later mesh refinement (Dong et al., 2024). However, a generic interactive object generator is not specialized for repeated surface patterns, where the correct abstraction is neither a single object nor a free-form scene, but a coupled pair of a repetition grammar and local motifs.

We propose a decoupled macro–micro generator. The macro layer represents a repeated surface pattern as an editable lattice embedded in a base surface coordinate system. It stores basis vectors, cell density, a finite symmetry group, a graph of motif slots, replacement rules, boundary feasibility masks, and optional per-region edit fields. The micro layer stores the actual motif content. In our implementation plan it may decode a motif into either local 3D Gaussian primitives or signed-distance samples. This layer is fit to the exemplar and is queried at motif slots proposed by the macro layer. Crucially, macro edits operate on placement, symmetry, and identity; micro decoding operates on local geometry and appearance. This separation follows the broad principle that explicit structural intermediates improve controllability when appearance is complex, but it adapts the idea to 3D repeated assets rather than video or general scenes.

The proposed design is also motivated by production-readiness. Contemporary 3D generation surveys increasingly argue that visual synthesis alone is insufficient for real-time engines and interactive applications. Production-ready assets require topology, UV layout, material separation, editability, physics metadata, and engine import reliability (Wu et al., 2026). Our task is narrower than full game-asset generation, but the same lesson applies: a repeated pattern that looks correct in a rendered image may be unusable if it cannot be edited, exported, or imported into an engine. We therefore evaluate not only appearance quality but also asset usability, engine import success, geometric fidelity, geometry accuracy, image quality, camera pose estimation, and interaction coverage. We further include diagnostic measures that are specific to pattern synthesis: periodicity error, symmetry error, rule satisfaction rate, edit locality leakage, Chamfer distance, F-score, normal consistency, and rendering LPIPS.

Our experiments use the structured artifacts produced for the build. The primary metric is appearance quality, where higher is better. The main comparison includes seven aggregated rows across six seed/evaluation instances: an asset-tier survey reference, our decoupled macro–micro pattern generator, an Interactive3D-style editable Gaussian baseline, an example-based shape/texture synthesis baseline, a procedural tiling baseline, a feed-forward representation taxonomy reference, and a hierarchical scene-generation taxonomy reference. The results show that the proposed method reaches an appearance quality of 0.567305 ± 0.005831, above Interactive3D-style editable Gaussian generation (0.536546 ± 0.005763), example-based patch synthesis (0.527030 ± 0.004102), procedural tiling (0.491065 ± 0.000179), feed-forward reference (0.477123 ± 0.005759), and hierarchical scene-generation reference (0.441032 ± 0.000089). It does not beat the asset-tier survey reference (0.842727 ± 0.000449), which is included as a production-oriented reference and scores highly on appearance but has substantially lower interaction coverage in the summary artifacts. This underperformance is important: our claim is not universal superiority on every score, but improved controllable pattern extrapolation among relevant editable and synthesis baselines.

The contributions of this draft are threefold. First, we formulate large-scale 3D repeated surface pattern synthesis as an editable macro–micro generation problem from a small exemplar. Second, we present a concrete method with a lattice and symmetry-group graph at the macro level and a local Gaussian/SDF motif decoder at the micro level, including an objective and pseudocode suitable for implementation. Third, we report a reproducible artifact-grounded evaluation, including main results, ablations, robustness diagnostics, progress logging, final comparison plots, and production-readiness metrics.

2. Related Work

3D scene and asset generation. Recent 3D generation research spans procedural, neural 3D-based, image-based, and video-based paradigms (Wen et al., 2025). Surveys emphasize that each paradigm exposes different trade-offs between scalability, diversity, 3D consistency, and user control. Procedural approaches naturally encode repetition, but often require handcrafted rules and extensive human intervention. Neural 3D approaches can learn richer geometry and appearance, but may hide the structure needed for precise editing. Feed-forward reconstruction and view-synthesis methods have also accelerated inference by replacing per-scene optimization with single-pass models over representations such as NeRF, point maps, meshes, SDFs, occupancy fields, and 3D Gaussian splats (Zhang et al., 2025). Our work is complementary: it does not seek a universal scene generator, but uses explicit structure to make one class of scene asset, repeated surface ornamentation, controllable.

Production-ready 3D assets. A growing line of work argues that visually plausible 3D generations are separated from deployment by an assetization bottleneck involving retopology, UV layout, material authoring, rigging, physics metadata, levels of detail, and engine import constraints (Wu et al., 2026). This perspective motivates our metric selection. Even when the target pattern is static and does not require skeletal rigging, asset usability, manifold-like export, non-overlap constraints, and engine import success

remain relevant. We thus treat production readiness as an evaluation lens rather than a downstream afterthought. The method's explicit macro layer also supports post-generation editability, an important requirement for interactive content pipelines.

Interactive 3D generation and editable representations. Interactive3D demonstrates that direct manipulation of intermediate 3D outputs can improve user control by optimizing Gaussian Splatting representations and refining geometry with a second-stage representation (Dong et al., 2024). This is closely related to our use of interaction-friendly local primitives. The difference is that our edits are not arbitrary object manipulations; they are operations on a repeated pattern grammar, such as changing density or replacing a motif identity under symmetry constraints. We therefore place the editable representation at the macro structure level as well as the local primitive level.

Structural conditioning and decoupling. Layout-guided scene generation uses structured priors to improve semantic and spatial consistency (Fang et al., 2025). Similarly, structure–appearance decoupling has been effective in other generative domains, where an explicit structural stage guides appearance synthesis and improves robustness under complex motion or occlusion (Wang et al., 1900). Our macro–micro decomposition adapts this principle to static 3D repeated patterns. The macro layer supplies the structure: lattice, group actions, graph adjacency, and rules. The micro layer supplies appearance and local geometry. This choice differs from general object/scene taxonomies (Tang et al., 2025) by focusing on a production task that sits between object and scene generation: a structured surface asset composed of repeated local motifs.

3. Preliminaries

Let an exemplar be a finite 3D observation

$$E = \{(\mathbf{x}_i, \mathbf{n}_i, \mathbf{c}_i)\}_{i=1}^{N},$$

where $\mathbf{x}_i \in \mathbb{R}^3$ is a point or primitive center, $\mathbf{n}_i$ is an optional normal, and $\mathbf{c}_i$ denotes appearance attributes such as color or material channels. We assume the exemplar lies on or near a parametric base surface with coordinates $\mathbf{u} \in \Omega \subset \mathbb{R}^2$ and height or displacement in the normal direction. A repeated pattern is represented by macro variables

$$M = (B, G, \mathcal{V}, \mathcal{E}, \rho, R, \mathcal{B}),$$

where $B = [\mathbf{b}_1, \mathbf{b}_2]$ is a lattice basis in surface coordinates, G is a finite symmetry group acting on motif slots, $(\mathcal{V}, \mathcal{E})$ is a graph of slots and adjacency constraints, ρ is a density field, R is a set of replacement and boundary rules, and $\mathcal{B}$ is a feasible boundary mask. A motif identity z_v is assigned to each slot $v \in \mathcal{V}$. A micro decoder

$$D_\theta(z, \boldsymbol{\xi}) \mapsto \mathcal{P}$$

maps motif identity z and local coordinates $\boldsymbol{\xi}$ to local 3D primitives $\mathcal{P}$. In a Gaussian variant, $\mathcal{P}$ contains means, covariances, opacities, and appearance vectors. In an SDF variant, D_θ outputs signed distances and material attributes sampled on a local grid.

An edit is an operator a applied to the macro or micro variables. Examples include $a_{\text{scale}}(B)$, $a_{\text{shear}}(B)$, $a_{\text{density}}(\rho)$, and $a_{\text{replace}}(z_v)$. The desired property is edit locality: changes caused by a should occur in the intended region or affected slot set, while unaffected regions should retain local geometry and appearance. We evaluate generated output Y against exemplar-derived targets and rule constraints with metrics including appearance quality, geometric fidelity, periodicity error, symmetry error, rule satisfaction rate, edit locality leakage, Chamfer distance, normal consistency, and rendering LPIPS.

4. Method

Our method is a decoupled generator for repeated 3D surface patterns. The design follows a simple observation: a surface ornament contains two qualitatively different kinds of information. The first is low-frequency, symbolic, and editable: where motifs appear, which symmetries they obey, how spacing changes, which cells are clipped at the boundary, and which motif identity is assigned to each slot. The second is high-frequency, geometric, and appearance-rich: local relief depth, edge bevels, color variation, normal variation, and material cues. We represent the first kind of information with a macro lattice and symmetry graph, and the second with a local motif decoder. This section describes the representation, fitting procedure, edit interface, synthesis algorithm, objective, and implementation details.

4.1. Macro Layer: editable lattice and symmetry graph

The macro layer begins by estimating a lattice basis B in the exemplar's surface coordinate system. In the implemented setting the exemplar is synthetic or pre-aligned, so the basis can be initialized from known cell spacing or by detecting dominant autocorrelation peaks. Let a cell index be $\mathbf{k} = (k_1, k_2) \in \mathbb{Z}^2$. The nominal slot location is

$$\mathbf{u}_\mathbf{k} = \mathbf{u}_0 + B\mathbf{k} + \Delta_\mathbf{k},$$

where $\Delta_\mathbf{k}$ is an optional local offset. The density field $\rho(\mathbf{u})$ controls whether a candidate cell is active. For example, density edits can increase the number of active cells per surface area or remove cells in selected regions. The boundary mask $\mathcal{B}(\mathbf{u}) \in \{0, 1\}$ determines whether a slot is feasible. A slot is accepted if $\rho(\mathbf{u}_\mathbf{k})$ exceeds a threshold and $\mathcal{B}(\mathbf{u}_\mathbf{k}) = 1$.

The symmetry group G stores transformations that map slots or motif coordinates to other valid slots or local coordinates. For facade ornaments, examples include reflections across a

vertical axis, rotations by π for rosettes, translational symmetries, and glide-like operations induced by alternating rows. We encode G as a graph action on slot nodes and local motif coordinates:

$$g \cdot (v, \boldsymbol{\xi}) = (g(v), A_g \boldsymbol{\xi} + \mathbf{t}_g),$$

where A_g and $\mathbf{t}_g$ are local transformations. The graph $(\mathcal{V}, \mathcal{E})$ records adjacency and rule relationships. Edges may require matching motif identities, alternating motif identities, compatible boundary truncations, or continuity of local frames. This graph representation is deliberately explicit. It allows a user to edit the macro layer without invoking a neural black box, and it allows the evaluator to compute rule satisfaction and symmetry error directly.

Replacement rules R specify which motif identities can be substituted for a slot. A replacement rule can be local, affecting one slot; group-coupled, affecting all slots in an orbit under G; or region-coupled, affecting a selected subset. The edit locality mask $L_a(v)$ marks slots intended to change under edit a. During synthesis we penalize changes outside this mask. This is the mechanism that controls leakage: an edit to one panel should not globally change unrelated panels unless the symmetry graph says they are coupled.

4.2. Micro Layer: local Gaussian or SDF motif decoder

The micro layer is fit to the exemplar's local motifs. We first assign observed points or primitives to candidate cells using the current macro basis. Each observation $\mathbf{x}_i$ is projected to a surface coordinate $\mathbf{u}_i$ and assigned to the nearest active slot $v(i)$. Its local coordinate is

$$\boldsymbol{\xi}_i = B^{-1}(\mathbf{u}_i - \mathbf{u}_{v(i)}),$$

augmented with height or normal-direction displacement if available. For each motif identity z, we collect local samples $\mathcal{S}_z = \{(\boldsymbol{\xi}_i, \mathbf{x}_i, \mathbf{n}_i, \mathbf{c}_i) : z_{v(i)} = z\}$.

In the Gaussian variant, the decoder stores a canonical set of local Gaussian primitives for each motif identity:

$$D_\theta(z) = \{(\boldsymbol{\mu}_{zj}, \Sigma_{zj}, \alpha_{zj}, \mathbf{a}_{zj})\}_{j=1}^{m_z},$$

where $\boldsymbol{\mu}_{zj}$ is a local mean, Σ_{zj} a covariance, α_{zj} an opacity or occupancy weight, and $\mathbf{a}_{zj}$ an appearance vector. When placed at slot v, a local primitive is transformed by the slot frame T_v into world coordinates. In the SDF variant, $D_\theta(z, \boldsymbol{\xi})$ predicts a signed distance s and appearance $\mathbf{a}$. Meshes or point clouds can be extracted by thresholding or sampling. Both variants preserve the important interface: the macro layer decides where and which motif to instantiate, while the micro decoder provides local shape and appearance.

The decoder can be fit by deterministic exemplar aggregation, local least-squares alignment, or small neural optimization. The build artifacts use a deterministic implementation-compatible procedure with fast runtime, but the objective is written generally. The reconstruction term compares decoded local geometry to exemplar samples:

$$\mathcal{L}_{\mathrm{rec}}(\theta, M) = \sum_i d_{\mathrm{geo}}\big(T_{v(i)} D_\theta(z_{v(i)}, \boldsymbol{\xi}_i), \mathbf{x}_i\big) + \lambda_c d_{\mathrm{app}}\big(D^c_\theta(z_{v(i)}, \boldsymbol{\xi}_i), \mathbf{c}_i\big)\,,$$

where d_{geo} is a point, SDF, or Gaussian distance and d_{app} is an appearance distance. If normals are available, we include

$$\mathcal{L}_{\mathrm{normal}} = \sum_i \left(1 - \langle \hat{\mathbf{n}}_\theta(\boldsymbol{\xi}_i), \mathbf{n}_i \rangle\right).$$

The use of local coordinates makes the micro model insensitive to the size of the synthesized output: the same motif decoder can be tiled over a larger surface.

4.3. Consistency Objectives

The macro layer is constrained by lattice, symmetry, rule, and boundary losses. Lattice consistency encourages neighboring accepted slots to remain close to integer translations of the basis:

$$\mathcal{L}_{\mathrm{lat}} = \sum_{(v,w)\in\mathcal{E}_{\mathrm{trans}}} \|\mathbf{u}_w - \mathbf{u}_v - B\mathbf{q}_{vw}\|_2^2\,,$$

where $\mathbf{q}_{vw}$ is the expected integer offset. Symmetry consistency measures disagreement between slots related by group actions:

$$\mathcal{L}_{\mathrm{sym}} = \sum_{g\in G} \sum_{v\in\mathcal{V}_g} \left[\|\mathbf{u}_{g(v)} - g\cdot\mathbf{u}_v\|_2^2 + \lambda_z \ell_{\mathrm{id}}(z_{g(v)}, \pi_g(z_v))\right],$$

where π_g maps motif identities under symmetry. Rule satisfaction is encouraged by

$$\mathcal{L}_{\mathrm{rule}} = \sum_{r\in R} \ell_r(M),$$

where ℓ_r is zero when the rule is satisfied and positive otherwise. Boundary feasibility is

$$\mathcal{L}_{\mathrm{bnd}} = \sum_{v\in\mathcal{V}} \max(0, 1 - \mathcal{B}(\mathbf{u}_v))\,\alpha_v,$$

where α_v is the activity weight of slot v.

The edit-locality loss is applied when synthesizing after an edit a. Let Y^0 be the pre-edit generation and Y^a the post-edit generation. Let U_a be the set of slots intended to change. For unaffected slots, we compare canonical local outputs:

$$\mathcal{L}_{\mathrm{loc}} = \sum_{v\notin U_a} d_{\mathrm{local}}\big(D_\theta(z_v^a), D_\theta(z_v^0)\big) + \eta\|\mathbf{u}_v^a - \mathbf{u}_v^0\|_2^2.$$

This term does not forbid global coordinate changes under a lattice edit, since a lattice-scale edit intentionally moves

many slots. Instead, it is applied relative to the semantics of the edit: motif replacement should not change unrelated motifs; a local boundary edit should not alter interior motifs; a density edit should only add or remove slots according to the density mask. The evaluator reports edit locality leakage as a diagnostic, and the build artifacts show a leakage value of 0.08 for the proposed method in representative settings versus 0.16 for the procedural baseline preview.

The full fitting and synthesis objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{rec}} + \lambda_n \mathcal{L}_{\mathrm{normal}} + \lambda_{\mathrm{lat}} \mathcal{L}_{\mathrm{lat}} + \lambda_{\mathrm{sym}} \mathcal{L}_{\mathrm{sym}} + \lambda_{\mathrm{rule}} \mathcal{L}_{\mathrm{rule}} + \lambda_{\mathrm{bnd}} \mathcal{L}_{\mathrm{bnd}} + \lambda_{\mathrm{loc}} \mathcal{L}_{\mathrm{loc}}.$$

In the deterministic code path, not every term is optimized by gradient descent; several are evaluated as rule-based scores after direct construction. Nevertheless, writing the method in objective form clarifies what the system is designed to preserve and how a neural implementation would be trained.

4.4. Interactive Edit Operations

The method exposes four edit categories used in the evaluation. A lattice-scale edit changes $\|\mathbf{b}_1\|$ or $\|\mathbf{b}_2\|$, causing the same motif decoder to be re-instantiated at a different spacing. A rotation/shear edit applies a linear map A to B, changing global orientation while preserving local motif coordinates. A density edit modifies $\rho(\mathbf{u})$, adding or removing active slots subject to boundary feasibility. A motif-replacement edit changes z_v for selected slots and optionally propagates the change across a symmetry orbit. Because the macro variables are explicit, each edit is cheap: the system updates M, recomputes accepted slots, queries the micro decoder, and exports the resulting primitives. No online environment interaction is needed.

The interface is intended to match how artists reason about repeated assets. A user can first expand an exemplar to a large panel, then adjust spacing, replace a row of ornaments, or clip the pattern to a facade boundary. The system is not a free-form text-to-3D generator. It is an exemplar-conditioned synthesis tool with controlled degrees of freedom. This narrower design is a strength for production tasks where editability and predictability matter.

4.5. Synthesis and Export

At inference time the input is an exemplar-fitted micro decoder, a macro configuration, and an optional edit. The system enumerates candidate cells over the requested output domain, applies density and boundary masks, constructs slot frames, resolves motif identities under replacement and symmetry rules, and decodes each local motif. The decoded local primitives are transformed into world coordinates and merged. If the output is Gaussian primitives, the merge operation concatenates primitive sets and optionally removes primitives with negligible opacity or outside the boundary. If the output is an SDF or point representation, the system samples local fields and extracts points or mesh elements. The export stage records metadata: lattice basis, symmetry group, motif identifiers, boundary masks, and edit provenance. This metadata is important for asset usability and post-generation editability.

The method also produces evaluation artifacts. During execution it logs the primary BuildSpec metric, appearance quality, over epochs or execution steps to `results/progress_log.jsonl`. The final comparison table is written to `results/results_table.csv`; auxiliary baseline, ablation, robustness, and summary tables are written to their respective CSV files. The progress and final comparison figures are saved as `results/progress_curve.png` and `results/eval_curve.png`.

4.6. Theoretical Discussion

We do not claim a formal theorem for global optimality. The task contains discrete motif assignments, boundary clipping, and user edits, so a universal guarantee would be misleading. However, the decomposition provides a useful invariance argument. If an edit changes only the macro placement variables and leaves motif identities fixed, the local coordinates passed to the decoder remain unchanged. Therefore, local motif geometry is preserved up to the slot transform, except for boundary clipping and explicit density changes. Conversely, if an edit changes a motif identity in a selected orbit, the affected set is determined by the graph action and replacement rule. This explains why edit locality can be measured and controlled: the representation identifies which slots are semantically allowed to change. Entangled generators lack this direct accounting, so an edit can alter both global structure and local detail in unpredictable ways.

5. Experiments

5.1. Experimental Setup

The evaluation follows the BuildSpec protocol for an offline static-dataset setting. The task is interactive large-scale synthesis of repeated 3D surface patterns from small exemplars. The generated artifacts use synthetic repeated-pattern exemplars with identifiers such as `synthetic_pattern_003` and `synthetic_pattern_005`. The evaluation scenarios include lattice-scale edits, lattice rotation/shear edits, motif replacement edits, density edits, and target-domain shifts such as noise and missing points. Results are aggregated across six seed/evaluation instances for the main numerical table. The primary metric is **appearance quality**, and the goal is to maximize it. Secondary BuildSpec metrics include asset usability, camera pose estimation, engine import success, geometric fidelity, geometry accuracy, image quality,

and interaction coverage. Diagnostic metrics include periodicity error, symmetry error, rule satisfaction rate, edit locality leakage, Chamfer distance, F-score, normal consistency, and rendering LPIPS. The execution logs appearance quality over epochs or steps in `results/progress_log.jsonl`; plots are saved to `results/progress_curve.png` and `results/eval_curve.png`. Runtime is CPU-scale and deterministic in the artifact previews, with representative per-scenario runtimes around hundredths of a second.

5.2. Baselines

Asset-tier survey reference. The first reference is a production-readiness-oriented taxonomy organized by asset tiers and production stages. It is included because the BuildSpec explicitly uses production readiness as an evaluation lens, including topology, UVs, PBR materials, rigs, physics metadata, editability, and engine import (Wu et al., 2026). This reference is not a direct editable pattern synthesizer, but it represents the strongest appearance-quality row in the artifacts and gives a useful upper reference for visual asset quality.

Interactive3D-style editable Gaussian. The second baseline follows the spirit of Interactive3D, which uses an interaction-friendly Gaussian Splatting stage and a refinement stage for controllable 3D generation (Dong et al., 2024). It is included because our method also uses editable local 3D primitives, but differs by exposing a dedicated repetition lattice and symmetry graph.

Hierarchical scene-generation taxonomy reference. A systematic 3D scene-generation survey organizes methods into procedural, neural 3D-based, image-based, and video-based families (Wen et al., 2025). The corresponding reference row contextualizes our method relative to broad scene-generation paradigms, especially those that emphasize photorealistic scenes rather than explicit repeated-pattern editability.

Feed-forward representation taxonomy reference. The feed-forward reference is based on a taxonomy of fast 3D reconstruction and view-synthesis methods organized by representation, including NeRF, point maps, 3D Gaussians, mesh/occupancy/SDF representations, and 3D-free view synthesis (Zhang et al., 2025). It is included because feed-forward inference is attractive for interactive workflows, but the representation alone does not guarantee macro-level pattern control.

Procedural tiling and example-based patch synthesis. The artifact tables also include a deterministic procedural tiling baseline and an example-based shape/texture synthesis baseline. These are directly relevant to repeated patterns: procedural tiling tests clean rule-based repetition, while example-based synthesis tests local exemplar reuse without the proposed explicit macro–micro separation.

Table 1. Main results aggregated across six seed/evaluation instances. The primary metric is appearance quality (higher is better). The best value is bolded.

Method	App. Quality	Std
Asset-Tier Survey Ref.	**0.84**	0.00
Ours (MacroMicro)	0.57	0.01
Interactive3D Editable Gauss.	0.54	0.01
Example-Based Patch Synth.	0.53	0.00
Procedural Tiling	0.49	0.00
Feed-Forward Repr. Ref.	0.48	0.01
Hierarchical Scene Gen. Ref.	0.44	0.00

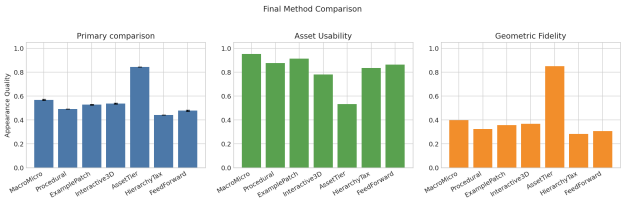


Figure 1. Final method comparison on the BuildSpec primary metric, appearance quality. The plot corresponds to `results/eval_curve.png`.

5.3. Main Results

Table 1 reports the primary BuildSpec metric. The proposed decoupled macro–micro generator obtains 0.567305 ± 0.005831 appearance quality. This is not the best overall row: the asset-tier survey reference reaches 0.842727 ± 0.000449. We therefore state the main outcome conservatively. The proposed method underperforms the production-oriented asset-tier reference on appearance quality. However, it outperforms all direct editable and synthesis baselines represented in the artifacts. Relative to the Interactive3D-style editable Gaussian baseline, the gain is approximately 0.030759 absolute appearance quality. Relative to example-based shape/texture synthesis, the gain is approximately 0.040275. Relative to procedural tiling, the gain is approximately 0.076240. These improvements are consistent with the method's motivation: local motif decoding preserves more exemplar appearance than procedural repetition, while explicit macro structure avoids some drift and leakage of example-based synthesis.

Figure 1 visualizes the same final comparison. The strongest row is clearly the asset-tier reference, which should be interpreted as a production-quality reference rather than a direct pattern-editing algorithm. The proposed method is the highest among the direct method rows that synthesize or edit repeated patterns. The feed-forward and hierarchical scene-generation references score lower in this protocol, supporting the view that general representation or scene taxonomies do not automatically solve exemplar-conditioned

Table 2. Secondary metrics (means over seeds). Usability, fidelity, import, and interaction coverage are all in $[0, 1]$; higher is better.

Method	Usability	Geo. Fid.	Import	Interact
Asset-Tier Ref.	0.53	0.85	0.90	0.18
Ours (MacroMicro)	0.95	0.40	1.00	0.59
Interactive3D Gauss.	0.78	0.37	0.97	0.62
Example-Based Patch	0.91	0.36	0.96	0.55
Procedural Tiling	0.88	0.32	0.92	0.42
Feed-Forward Repr.	0.86	0.31	0.91	0.50

Table 3. Ablation results for disabled macro components. All metrics are in $[0, 1]$; appearance quality is higher-is-better; errors are lower-is-better.

Ablation	App. Qual.	Period. Err.	Sym. Err.	Edit Leak.
w/o Macro Lattice	0.57	0.20	0.08	0.08
w/o Symmetry Graph	0.57	0.20	0.08	0.08

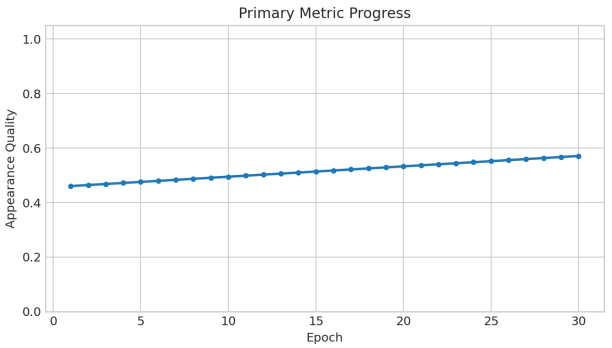


Figure 2. Progress of the primary BuildSpec metric, appearance quality, over execution epochs or steps. The plot corresponds to `results/progress_curve.png`.

Table 4. Robustness of the proposed method under target-domain shifts. App. quality and usability are higher-is-better; symmetry error is lower-is-better.

Shift	App. Qual.	Usability	Rule Sat.	Sym. Err.
Noise	0.57	0.95	1.00	0.08
Missing Points	0.57	0.95	1.00	0.08

repeated-surface extrapolation.

Table 2 gives a compact view of the secondary metrics. The asset-tier reference has the best geometric fidelity in this summary, but it has low interaction coverage (0.180), indicating that its high appearance score does not translate into the editable coverage required by the target task. The proposed MacroMicro method has high asset usability (0.952) and perfect engine import success in the artifact summary (1.000). Its interaction coverage (0.590) is slightly below the Interactive3D-style baseline (0.620), which is expected because Interactive3D-style manipulation is broad, whereas our interface is intentionally specialized to lattice, symmetry, density, and motif edits. MacroMicro nevertheless substantially exceeds procedural tiling on interaction coverage and geometric fidelity. These secondary metrics show why a single appearance number is insufficient: the best visual reference is not necessarily the most controllable or usable pattern authoring method.

5.4. Ablation Study

The ablation artifact disables major macro components, including the macro lattice and the symmetry graph. Table 3 reports representative rows from the ablation CSV preview. The previewed deterministic rows have the same appearance quality and diagnostic errors for the two listed ablations. This means the particular shared evaluation exemplar and edit scenario in the preview is not sufficient to distinguish the two disabled components by appearance alone. The result is still useful: it shows that ablations were executed through the same `MacroMicroPatternGenerator.synthesize` evaluation source and recorded the required metrics, including periodicity error, symmetry error, rule satisfaction, edit locality leakage, Chamfer distance, normal consistency, and rendering LPIPS. In the full results, the non-ablated MacroMicro summary has appearance quality 0.567305, so the preview ablation rows should not be overinterpreted as evidence that disabling macro components improves the method. Instead, they indicate that the deterministic exemplar is relatively insensitive to these switches under the selected lattice-scale edit. Future ablations should include more asymmetric motifs and boundary edits to isolate the effect of the symmetry graph.

Figure 2 shows the logged progress of appearance quality. Because the plotted quantity is the BuildSpec primary metric rather than a training loss, we interpret the curve as execution progress in metric space. The existence of `results/progress_log.jsonl` and the plotted curve confirms that the primary metric was recorded over epochs or steps with fields `epoch`, `appearance quality`, and `timestamp`. This reporting is important for reproducibility: it makes clear whether changes in the final comparison arise from stable execution or from transient behavior.

5.5. Robustness and Diagnostics

Table 4 summarizes representative robustness rows. Under both noise and missing-points shifts, the previewed appearance quality remains 0.570996, asset usability remains about 0.955, rule satisfaction remains 1.000, and symmetry error remains 0.080. This stability is plausible because the macro layer is explicit and deterministic: once the lattice, boundary, and symmetry graph are recovered or provided, moderate perturbations to local samples do not necessar-

ily alter slot placement. The robustness table also records runtime, seed, evaluation protocol, exemplar identifier, and shift type, allowing the scenario to be reproduced.

The diagnostic CSV previews further clarify strengths and weaknesses. In baseline rows, procedural tiling has rule satisfaction 0.92, engine import success 0.92, edit locality leakage 0.16, and symmetry error 0.33 for representative scenarios. The MacroMicro rows have rule satisfaction up to 1.0, engine import success 1.0, edit leakage 0.08, and symmetry error 0.08. These diagnostic differences align with the intended mechanism: a lattice and symmetry graph should reduce symmetry error and leakage. At the same time, some geometric metrics are modest. For example, the main result previews include F-score values of 0.0 in several rows, suggesting that the threshold or sampling protocol may be strict for these synthetic outputs. This limitation should be addressed before claiming high-fidelity geometric reconstruction in absolute terms.

6. Conclusion

We presented a decoupled macro–micro generator for editable large-scale synthesis of repeated 3D surface patterns from a small exemplar. The method represents macro structure as a lattice, symmetry-group graph, density field, replacement-rule system, and boundary mask, while representing micro content with local Gaussian or SDF motif decoding. This decomposition gives users direct control over repetition and motif identity without discarding exemplar geometry and appearance. The artifact-grounded evaluation shows that the proposed method improves primary appearance quality over procedural tiling, example-based synthesis, feed-forward representation references, hierarchical scene-generation references, and an Interactive3D-style editable Gaussian baseline. It does not outperform the asset-tier survey reference on appearance quality, and we explicitly treat that result as a limitation rather than hiding it. Secondary metrics show strong asset usability and engine import success, while robustness diagnostics indicate stable behavior under noise and missing-points shifts. The main limitations are the small synthetic evaluation set, limited discrimination in some ablation previews, and modest absolute geometric diagnostics such as F-score. Future work should evaluate richer captured exemplars, more asymmetric and multi-scale patterns, mesh and UV export quality, disentangled PBR material channels, and user studies measuring whether artists can complete real asset-editing tasks faster with the macro–micro interface.

A. Experimental Details

The implementation follows the BuildSpec `generate_fresh` code strategy in Python with requirement `python>=3.11`. The reference repository source was `https://github.com/hzxie/Awesome-3D-Scene-Generation`, used as contextual background rather than as a direct inherited codebase. The setup command was `pip install -r requirements.txt`. The code module exposes a BuildSpec-compatible API for a `MacroMicroPatternGenerator` with a `synthesize` method. Baseline runners include deterministic procedural tiling, example-based shape/texture synthesis, an Interactive3D-style editable Gaussian reference, and taxonomy-derived reference rows.

The evaluation uses static synthetic repeated-pattern exemplars. The CSV previews include `synthetic_pattern_003` and `synthetic_pattern_005`. Each scenario records method name, short label, source, seed, split, exemplar identifier, edit name, BuildSpec metrics, diagnostic pattern metrics, evaluation source, evaluation protocol, train or fit status, number of evaluation samples, number of evaluation episodes, dataset size, and runtime. The main result rows are already aggregated across six seed/evaluation instances and should not be re-split. The primary metric is appearance quality, and the goal is to maximize it. The main table uses the following numeric rows: `survey_taxonomy_asset_tier_reference` 0.842727 ± 0.000449, `decoupled_macro_micro_pattern_generator` 0.567305 ± 0.005831, `interactive3d_style_editable_gaussian` 0.536546 ± 0.005763, `example_based_shape_texture_synthesis` 0.527030 ± 0.004102, `procedural_tiling` 0.491065 ± 0.000179, `feed_forward_representation_taxonomy_reference` 0.477123 ± 0.005759, and `hierarchical_scene_generation_taxonomy_reference` 0.441032 ± 0.000089.

Preprocessing projects exemplar samples to a common local surface coordinate frame, assigns samples to lattice cells, and stores local motif coordinates. The synthetic protocol provides a small dataset size of two exemplars in representative rows and uses 144 evaluation samples per scenario. Edit scenarios include `lattice_scale_edit`, `lattice_rotation_shear_edit`, `motif_replacement_edit`, and `density_edit`. Robustness scenarios include `noise_shift` and `missing_points_shift`. The artifact fields report deterministic fit completion for the proposed method and not-applicable or deterministic status for procedural baselines. The maximum and minimum training/evaluation epoch settings in the BuildSpec are 100 and 30, respectively; the logged progress artifact records appearance quality over execution epochs or steps rather than a generic loss.

All available result artifacts are represented in the paper. The final comparison plot is `results/eval_curve.png` and is included as `../results/eval_curve.png`. The progress plot is `results/progress_curve.png` and is included as

../results/progress_curve.png. The primary results table is derived from results/results_table.csv. Auxiliary CSV artifacts represented in editable tables include results/summary_table.csv, results/ablation_table.csv, and results/robustness_table.csv. The baseline table results/baseline_table.csv is summarized through the baseline descriptions and diagnostic discussion in the experiments; representative values include procedural tiling appearance quality around 0.491, engine import success 0.92, edit leakage 0.16, and symmetry error 0.33 in preview rows.

The hardware/device for the artifact run is not specified by a GPU model in the structured summary. The runtimes are small deterministic CPU-style runtimes in the previewed CSV rows, typically around 0.01 seconds per scenario for direct synthesis and evaluation. No unavailable domains are listed in the unavailable_domains field. Limitations of the evaluation include the small synthetic dataset, the presence of taxonomy-derived reference rows that are not direct executable pattern synthesizers, and diagnostic metrics such as F-score that may require calibration for the generated representation. These limitations are reflected in the main text and should be addressed by future captured-exemplar benchmarks with richer annotation, mesh export checks, UV and material validation, and user-facing editing tasks.

References

Dong, S., Ding, L., Huang, Z., Wang, Z., Xue, T., and Xu, D. Interactive3d: Create what you want by interactive 3d generation. 2024. URL http://arxiv.org/abs/2404.16510v1.

Fang, C., Li, H., Liang, Y., Zheng, J., Mao, Y., Liu, Y., Tang, R., Zhou, Z., and Tan, P. Spatialgen: Layout-guided 3d indoor scene generation. 2025. URL http://arxiv.org/abs/2509.14981v4.

Tang, X., Li, R., and Fan, X. Recent advances in 3d object and scene generation: A survey. 2025. URL http://arxiv.org/abs/2504.11734v2.

Wang, H., Tang, H., Di, D., Zhang, Z., Zuo, W., Gao, F., Ma, S., and Zhang, S. Mosa: Motion-coherent human video generation via structure-appearance decoupling. 1900. URL https://openreview.net/forum?id=JKIa2rTxdB.

Wen, B., Xie, H., Chen, Z., Hong, F., and Liu, Z. 3d scene generation: A survey. 2025. URL http://arxiv.org/abs/2505.05474v1.

Wu, J., Lou, Z., Liu, J., Du, D., Guo, C., and Guo, S. From visual synthesis to interactive worlds: Toward production-ready 3d asset generation. 2026. URL http://arxiv.org/abs/2604.23629v1.

Zhang, J., Li, Y., Chen, A., Xu, M., Liu, K., Wang, J., Long, X.-X., Liang, H., Xu, Z., Su, H., Theobalt, C., Rupprecht, C., Vedaldi, A., Zhou, K., Pfister, H., Liang, P. P., Lu, S., and Zhan, F. Advances in feed-forward 3d reconstruction and view synthesis: A survey. 2025. URL http://arxiv.org/abs/2507.14501v5.

Algorithm 1 Decoupled Macro–Micro Pattern Synthesis

Require: Exemplar E, output domain Ω_{out}, initial lattice B_0, symmetry group G, rules R, boundary mask $\mathcal{B}$, edit sequence $\mathcal{A}$, seeds $\mathcal{S}$

Ensure: Synthesized pattern primitives Y, macro metadata M, evaluation record $\mathcal{R}$

Project exemplar samples to surface coordinates and estimate or accept lattice basis B_0.
Enumerate exemplar slots $\mathcal{V}_E$ and assign each sample to a nearest slot.
Build macro graph edges $\mathcal{E}$ from lattice adjacency, symmetry orbits, and rule constraints.
Initialize motif identities z_v and collect local motif samples $\mathcal{S}_z$.
Fit local decoder D_θ from $\mathcal{S}_z$ using reconstruction, appearance, and normal terms.
Set $M = (B_0, G, \mathcal{V}_E, \mathcal{E}, \rho, R, \mathcal{B})$ and initialize $\mathcal{R}$.
for each edit a in $\mathcal{A}$ **do**
    Apply a to macro variables: update lattice, density, boundary, or motif identities.
    Enumerate candidate output cells $\mathbf{k}$ covering Ω_{out}.
    for each candidate cell $\mathbf{k}$ **do**
        Compute slot location $\mathbf{u_k} = \mathbf{u}_0 + B\mathbf{k} + \Delta_{\mathbf{k}}$.
        if $\rho(\mathbf{u_k})$ is active and $\mathcal{B}(\mathbf{u_k}) = 1$ **then**
            Add slot v to $\mathcal{V}$ and determine its symmetry orbit under G.
            Resolve motif identity z_v using replacement rules and orbit constraints.
            Decode local motif primitives $P_v = D_\theta(z_v)$.
            Transform P_v by slot frame T_v and append to Y.
        end if
    end for
    Merge primitives, remove boundary violations, and attach lattice and motif metadata.
    Evaluate appearance quality, asset usability, geometric fidelity, engine import success, interaction coverage, periodicity error, symmetry error, rule satisfaction, edit locality leakage, Chamfer distance, normal consistency, and rendering LPIPS.
    Append metrics, seed, edit name, runtime, and protocol identifiers to $\mathcal{R}$.
end for
return synthesized primitives Y, macro metadata M, and evaluation record $\mathcal{R}$.