Seamless Tileable Hexagonal Texture Generation with a Latent Diffusion Model Using Multi-Pass Inpainting and Latent Space Manipulation

Dhafin Fawwaz Ikramullah (*Author*)
Informatics Engineering Study Program School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
13522084@std.stei.itb.ac.id

***Abstract*—Digital textures should ideally be seamless and tileable so that a single small image can be repeated to cover a large surface without revealing any visible seam. Nearly every existing tileable texture synthesis method, from patch based approaches to GAN and diffusion based ones, is designed for square grids only, even though the hexagon also tiles the plane without gaps and is considerably more effective at hiding the perception of repetition. This paper proposes SeamlessHexDiffusion, a method for generating seamless tileable hexagonal textures on top of a pretrained latent diffusion model that requires no retraining for each texture. The method maps the input texture onto a hexagonal arrangement, casts the seam region as a two pass inpainting problem, and then enforces hexagonal periodicity through latent space manipulation consisting of periodic copying and random hexagonal shifting at every denoising step, with an IP Adapter carrying the style of the input texture. Evaluation covers 602 runs, namely 43 configurations over 14 textures, using five metrics together with paired Wilcoxon tests. Tileability measured by TexTile rises from 0.437 for a texture tiled hexagonally without any diffusion to 0.581, with a p value of 0.030, at a reasonable cost in fidelity. Per texture finetuning and ControlNet are shown not to be worth their cost. Generating one texture takes about 40 seconds, far shorter than prior methods that mandate per texture training.**

Keywords—seamless tileable texture; hexagonal grid; latent diffusion model; multi-pass inpainting; latent space manipulation; IP Adapter; TexTiles

I. INTRODUCTION

Digital textures are an essential element across visual domains, from games and computer graphics to architecture, fashion, and manufacturing. They are used to realistically represent material surfaces such as fabric patterns, batik, ceramic motifs, wood, and metal. To cover large surfaces without consuming large amounts of memory, a texture is ideally made tileable, meaning it can be repeated side by side, and seamless, meaning no visible boundary appears between adjacent copies [1].

Producing seamless tileable textures by hand is time consuming and demands considerable artistic skill, which is why a range of computational methods has been developed to automate it. Early approaches were non parametric, for instance image quilting, which stitches together small patches of an image [2], graph cuts, which finds an optimal cut through the overlap region between patches [3], and texture splatting and texture bombing, which randomize a texture over a wide area but only work for stochastic textures [4]. Specifically for tileability, subsequent work introduced histogram preserving blending for patch based synthesis [5], texture stationarization, which maximizes stationarity [6], and a graph cuts based algorithm that reshapes patch borders so that they can be joined [7]. Deep learning based approaches followed, ranging from repeating pattern detection using pretrained CNN features [8] and self organizing textures built on Neural Cellular Automata [9] to PSGAN, which parameterizes the latent space periodically [10]. The state of the art GAN based method is SeamlessGAN [11], which tiles the latent space inside a generative network and then crops its central area to obtain a tileable texture, using the discriminator as a perceptual quality metric. Once diffusion models became dominant the field shifted accordingly, as in Tiled Diffusion, which imposes tiling constraints and similarity constraints on the latent space along with a round robin mechanism for many to many relations between tiles [12].

All of these methods share one assumption, namely that the tile is a square. A texture only needs to join along four sides, top with bottom and left with right. Yet only three regular polygons can cover the plane without gaps using a single shape, namely the triangle, the square, and the hexagon. The hexagonal case remains untouched by AI based texture synthesis, even though hexagonal adaptations are commonplace elsewhere. Hexagonal sampling theory is in fact more efficient than square sampling in signal processing [13], hexagonal convolution variants such as HexaConv [14] and hexagonal convolutions for hexagonal grids [15] exist, and in real time rendering hex tiling is already used to disguise texture repetition because hexagonal structure is harder for the eye to latch onto than a square grid [16]. Hexagonal grids also look more natural because they approximate patterns found in nature such as honeycombs, crystal structures, and scales.

Adapting existing methods to a hexagonal grid is far from trivial. A latent diffusion model [17] relies on a Variational

Autoencoder (VAE) [18] and a U-Net [19] that operate only on rectangular tensors, so a Tiled Diffusion style tiling constraint cannot be applied directly. The latent vector produced by the VAE encoder also cannot be shifted along oblique directions freely, because its spatial information is laid out on a square grid, and forcing such a shift introduces zigzag artifacts after decoding. Furthermore, a hexagon placed on a rectangular canvas leaves empty regions near the canvas corners that must be filled. The fundamental difference is the neighbor count, namely four for a square and six for a hexagon. This paper proposes SeamlessHexDiffusion, which resolves these issues. The contributions are as follows.

- A method for generating seamless tileable textures on a hexagonal grid on top of a pretrained latent diffusion model which, to the best of the author's knowledge, is the first for the hexagonal case, and which requires no per texture training.
- A formulation of hexagonal tileability as an inpainting problem, by way of a geometry mapping that produces a mapped texture together with a mask covering the seam region.
- Two hexagonal latent space operators, namely hexagonal periodization, which copies the hexagon interior into the region outside it so that the U-Net always sees an exactly tileable context, and random hexagonal shifting at each denoising step so that boundary cells take turns being processed as interior. The shift is designed as a permutation over the set of hexagon pixels, so it is exactly invertible and does not accumulate degradation over the course of denoising.
- Multi pass inpainting scheme with an IP Adapter [20] as the style carrier, which separates filling the exterior region from enforcing tileability inside the tile region.
- A thorough evaluation of 602 runs using five metrics, paired Wilcoxon tests, a sensitivity analysis over 14 hyperparameters, a computational cost analysis, and a comparison against seven prior methods, including two negative results that are useful for follow up work.

II. Related Work

A. Seamless Tileable Texture Synthesis

Tileability has received comparatively little attention next to texture synthesis in general. Moritz et al. proposed a non parametric approach that preserves texture stationarity [6], while Li et al. search for the most representative patch and then reshape its borders with graph cuts [7]. Deliot and Heitz use a histogram preserving blending operation for patch based synthesis [5]. In the deep learning setting, Rodriguez-Pardo et al. exploit the latent space of a pretrained network to find the size of the repeating pattern and then search for the optimal crop [8], Niklasson et al. generate textures that are tileable by construction using Neural Cellular Automata [9], and Bergmann et al. achieve tileability through a periodic spatial manifold in a GAN [10].

B. SeamlessGAN

SeamlessGAN [11] is the main reference for this work. Its key idea is that tiling the latent space inside a generative network produces outputs that are continuous at the seam intersection, so a tileable texture is obtained by cropping the central area. A source crop is encoded into a latent vector, its spatial resolution is tiled two by two and concatenated, and the result is passed through several residual blocks and two separate decoders for the albedo map and the normal map. Because not every latent value yields a high quality output, a trained PatchGAN discriminator is used as a perceptual error metric during a sampling procedure.

Two properties of SeamlessGAN motivate the present work. First, tileability is achieved through latent space manipulation rather than by altering the convolutional architecture, and that idea transfers to a latent diffusion model. Second, SeamlessGAN mandates per texture adversarial training that takes roughly 45 minutes, leaving considerable room for improvement in efficiency. On the other hand, the two by two latent tiling and central crop mechanism is inherently square specific and has no direct hexagonal counterpart.

C. Tiled Diffusion

Tiled Diffusion [12] shares latent space information during denoising through a tiling constraint, copying part of the right edge of the latent as padding on the left and vice versa, and likewise for the top and bottom edges. For the multi image case, a similarity constraint copies part of the latent between images, and a round robin mechanism ensures every edge is exposed to its partner in turn. The method demonstrates that intervening on the latent at each denoising step is sufficient to enforce tileability without changing model weights. All of its operations, however, are defined on the sides of a square.

D. Quality Metrics for Tileable Textures

Evaluating tileable textures requires a metric that can detect local discontinuities at tile boundaries, handle arbitrary dimensions and aspect ratios, and understand an image globally in order to spot artifacts [21]. TexTile [21] is a differentiable metric designed for exactly that purpose. It is implemented as a CNN based binary classifier that takes a texture tiled on a two by two grid as input and produces a tileability score between zero and one, where a higher score means the texture is more suitable for tiling. Its advantage over existing metrics is that it measures how repeatable a texture is explicitly.

On the fidelity side, DISTS [22] unifies structural similarity and texture statistics similarity over VGG16 features and is tolerant to resampling and local displacement. That property matters for a method that resynthesizes every pixel. SSIM [23], LPIPS [24], and SI-FID [25], [26] are also used so that results remain connected to the evaluation protocol used by SeamlessGAN.

III. Proposed Method

A. Overview

The central idea is to convert hexagonal tileability into a periodically controlled inpainting problem. Rather than forcing

the model to operate on hexagon shaped tensors, the input texture is first mapped onto a hexagonal arrangement on a rectangular canvas, so the model still sees an ordinary rectangular tensor, while hexagonal periodicity is enforced by intervening on the latent space at every denoising step.

The overall pipeline is shown in Fig. 1, organized along two axes. Horizontally it moves between pixel space, where geometry mapping and color correction take place, and latent space, where every denoising operation lives. Vertically it is divided into two inpainting passes. Pass one is a plain inpainting step that fills the exterior region, whereas pass two re-inpaints the hexagon interior together with latent vector manipulation. Geometry mapping and VAE encoding precede the first pass, and VAE decoding, inverse geometry mapping, and color correction close the second.

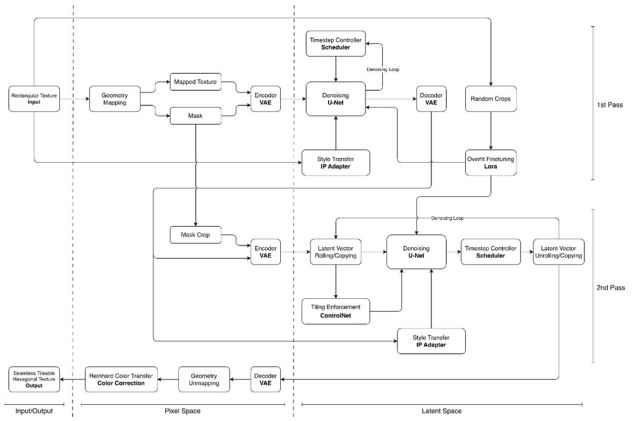


Fig. 1. Overview of SeamlessHexDiffusion. The columns separate input and output, pixel space, and latent space, while the rows separate the two inpainting passes. The first pass maps the square input texture onto the hexagonal lattice to produce a mapped texture and a mask, encodes both with the VAE, and fills the exterior region through a plain denoising loop conditioned by the IP Adapter. The second pass re-encodes the decoded first pass result together with the cropped mask and runs a second denoising loop in which the latent vector is rolled and copied hexagonally before each U-Net call and unrolled afterwards, which is what enforces hexagonal periodicity. The result is VAE decoded, inverse mapped, and color corrected. Random crops with LoRA finetuning and ControlNet tiling enforcement are shown for completeness but are disabled in the reference configuration.

The base model is runwayml/stable-diffusion-inpainting together with its bundled VAE, with DDIM as the sampling scheduler [27]. All model weights are frozen, so the method is training free in its reference configuration.

B. Hexagonal Geometry Mapping

Geometry mapping takes one input texture and produces three outputs, namely the texture rearranged into a hexagonal pattern, a mask marking the region the model must fill, and auxiliary information needed for the inverse mapping at the end. If the input is not square, its central region is cropped to a square and resized.

The process can be pictured as a camera looking at an arrangement of pointy top hexagons, with the camera positioned at the centroid between three hexagons. For a hexagon of circumradius R, the centers of two horizontally adjacent hexagons are separated by $\sqrt{3}R$, while the next row of centers sits $3R/2$ lower and is staggered horizontally by $\sqrt{3}R/2$. The set of all integer combinations of those two displacements forms the hexagonal lattice Λ, and an output texture is hexagonally seamless and tileable exactly when it is invariant under every translation in Λ. The rectangular unit cell of this lattice measures $\sqrt{3}R \times 3R$ and contains two hexagons, and this is the size used later when preparing input for the TexTile metric.

For every canvas pixel the nearest hexagon is found by converting the pixel position into hexagonal coordinates, rounding to the nearest cell, and converting back into a hexagon center position. Each pixel's texture coordinate is then computed relative to that nearest hexagon center, so the input texture repeats along the hexagonal pattern. Sampling is performed at low resolution and then upscaled with the LANCZOS algorithm, because sampling directly at high resolution coordinates introduces aliasing around pixel edges. The canvas size is rounded up to a multiple of eight as required by the VAE.

Two radii must be distinguished. The content radius is that of the hexagon actually filled with input texture, set by the hypotenuse hyperparameter. The camera radius is that of the lattice cell the model actually works on, and it is the radius that fixes Λ. The outer_margin hyperparameter is what separates the two. When it is zero the two coincide and every lattice cell is completely covered by texture. As it grows, the lattice period widens beyond the content hexagon and an empty ring opens up between them, and it is that ring the model must synthesize. A second hyperparameter, camera_padding, instead enlarges the working canvas around the same lattice cell, so it widens the context available to the model without altering either the lattice period or the tile content. This difference in role later explains why the two produce very different fidelity trade offs.

The mask is built from two signed distance functions computed relative to the nearest hexagon center. The first measures the inward distance from the edge of the content hexagon, and inner_padding sets how far inside that edge the mask still reaches, so that the model is allowed to rework a band along the tile boundary rather than only the empty ring. The second measures the inward distance from the intersection between the content hexagon and the square frame of the source texture, with gap_padding playing the same role for it; this second term is what covers the hexagon corners that a square source texture cannot reach. The final mask is the union of the two.

Fig. 2. Geometry mapping visualization. Hyperparameters (left), mapped texture (center), and mask (right).

C. Multi-Pass Inpainting

The region the model must fill consists of two parts with very different character. The exterior region, the ring outside the content hexagon, starts from an empty canvas and its task is to grow new texture whose style connects with its surroundings. The seam region inside the hexagon already contains texture and its task is to align opposing edges so that they can be tiled. Filling both at once forces a costly compromise.

The simultaneous mode fills the entire mask in one process and forces strength to one. The reason is mechanical. The geometry mapping output is black in the gap regions, and when strength is below one the denoising start point is the noised latent of the source texture, which therefore still carries that black information. The model is then continually pulled toward black in the gap regions and cannot escape it, especially since the text prompt used is empty. With strength set to one, denoising starts from pure noise and the problem disappears, but at the cost of discarding all texture structure inside the hexagon as well.

The two-pass mode separates the two jobs. Pass one fills the exterior region with rolling and ControlNet disabled. Two implementation variants are available, namely a scheduled variant that runs the same denoising loop as pass two but with both mechanisms turned off, and a built in variant that calls the standard inpainting pipeline. The IP Adapter may be disabled during pass one via use_on_pass1, implemented by temporarily storing the reference embedding and restoring it once pass one finishes. After pass one completes, the IP Adapter reference may be recomputed from its output.

The pass two mask is formed by clipping the original mask to the hexagon interior, but the clip is deliberately given a margin of roughly two latent cells so that the frozen cells end up genuinely outside the latent space hexagon. One latent cell spans eight pixels, so an overly tight clip would freeze cells that are partly still inside the hexagon and produce a hard edge along the tile boundary. If the clipped mask turns out to be entirely zero, for example because inner_padding and gap_padding are both zero so nothing remains to be done inside the hexagon, pass two is skipped and the pass one output is used directly.

The ControlNet reference texture for pass two is a composite of the two sources available at that point. Wherever the mask is zero it takes the geometry mapped texture, and wherever the mask is one it takes the pass one output, blending smoothly in between. The part still originating from the original input texture is preserved because pass one inevitably degrades quality slightly, while the part that was originally empty can only come from pass one

Pass two then reinpaints the hexagon interior using the pass one output as its source texture, with latent manipulation active and ControlNet optional. This separation also improves conditioning quality, since by pass two neither the IP Adapter nor the ControlNet reference texture contains empty regions any more.

D. Denoising Loop with Hexagonal Latent Space Manipulation

This is the main process and the site of most of the technical contribution. The underlying problem is that the U-Net cannot guarantee a tileable output, because latent cells at the hexagon boundary are always adjacent to the region outside the hexagon and never see their counterpart cells on the opposite side. The model has no way of knowing that those two edges will eventually touch.

The denoising loop is therefore written from scratch rather than calling the pipeline's built in call method, because each step requires six interventions that the standard interface does not expose, namely hexagonal periodization over every input tensor, exactly invertible hexagonal shifting, per step re-injection of the frozen region, manual ControlNet injection over already shifted tensors, nine channel input assembly performed after shifting, and restoration of tensor positions at the end of the step.

The working canvas is eight times finer than its latent, so the hexagon radius must be tracked at two scales. The latent space radius is one eighth of the pixel space radius, up to a small correction arising from rounding the canvas up to a multiple of eight. Every latent space tensor uses the latent radius, whereas the ControlNet reference texture, which stays at pixel resolution, uses the pixel radius.

Six tensors are prepared before the loop begins, namely the initial latent, the noise in use, the clean source texture latent, the latent mask obtained by downsampling the pixel mask, the latent of the texture with its masked region blanked, and the ControlNet reference texture. The timestep schedule comes from the scheduler, and when strength is below one a prefix of the steps is skipped so denoising begins from a texture that still retains part of its original information. For example, strength of 0.7 with 50 steps means only the final 35 steps are executed.

This operator overwrites every position outside the central hexagon with a copy of the hexagon interior, following the lattice Λ,

$$\Phi_R[z](\mathbf{p}) = \begin{cases} z(\mathbf{p}), & \mathbf{p} \in H(R) \\ z(\mathbf{p} - \mathbf{v}(\mathbf{p})), & \mathbf{p} \notin H(R) \end{cases} \qquad \mathbf{v}(\mathbf{p}) = \arg\min_{\mathbf{v} \in \Lambda} \|\mathbf{p} - \mathbf{v}\|$$

where H(R) is the hexagon of circumradius R at the canvas center, so that p − v (p) always lands inside it. In other words, the region outside the hexagon is turned into a periodic shadow of its interior, and the U-Net sees a context that is already exactly tileable. Without this operator the U-Net sees a discontinuity at the hexagon boundary, and that discontinuity surfaces as a seam in the final output.

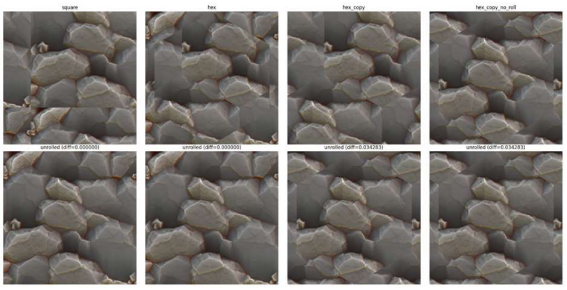
Two implementation details determine the resulting quality. First, because the hexagonal lattice is not aligned with the integer pixel grid, the source position p − v(p) almost always falls between pixels and must be rounded. That rounding occasionally lands just outside the hexagon, and if left unchecked the value copied is stale content from outside the hexagon, which then propagates further on the next invocation. Every source position falling outside the hexagon is therefore snapped to the nearest pixel inside it. Second, the operator is applied to every tensor the U-Net will see, not just the latent, namely the initial latent, the noise, the clean source latent, the masked texture latent, the latent mask, and the ControlNet reference. If any one of them is not periodic, the U-Net still receives a signal that breaks at the hexagon boundary.

Periodization alone is not enough. Boundary cells do see a shadow of their neighbors, but they remain boundary cells throughout denoising, so the model never treats them as interior content whose continuity must be maintained. At each step, therefore, every tensor is shifted by a random offset drawn from inside the hexagon, and the shift wraps modulo the lattice Λ rather than modulo the canvas rectangle. Content leaving one side reappears from the opposite side following the hexagonal repetition pattern, so material that was at the boundary moves toward the center and gets processed as interior. Some other region always takes its place at the boundary, but the effect is small because any given position is likely to be a boundary position for only a single step.

It is worth emphasizing that this is not an ordinary circular shift on a rectangular tensor. The difficulty is that folding a shifted position back into the hexagon does not yield an integer position, and naive rounding creates two problems at once. Some source positions fall outside the hexagon, and several destination positions compete for the same source pixel while other source pixels go unused entirely. The mapping is then no longer one to one and the operation cannot be inverted exactly.

The implementation resolves this in two stages. Source positions falling outside the hexagon are snapped to the nearest pixel inside it. Collisions are then resolved by giving priority to the destination position with the smallest rounding error, while losing destinations are reassigned to the nearest unclaimed pixel inside the hexagon. The result is a permutation over the set of hexagon pixels, so its inverse can be constructed exactly by inverting the mapping table. It is this permutation property that makes the sequence of shift, one scheduler step, then restore consistent across steps. Without that guarantee the latent would lose a small amount of information at every step, and that loss would accumulate over the course of denoising.

The offset is drawn by rejection sampling, taking a uniform integer pair over the hexagon's bounding box and rejecting both the zero offset and any value falling outside the central hexagon, so the offset is always a valid lattice displacement. The ControlNet reference texture is shifted by the equivalent offset in pixel scale, that is, eight times larger.



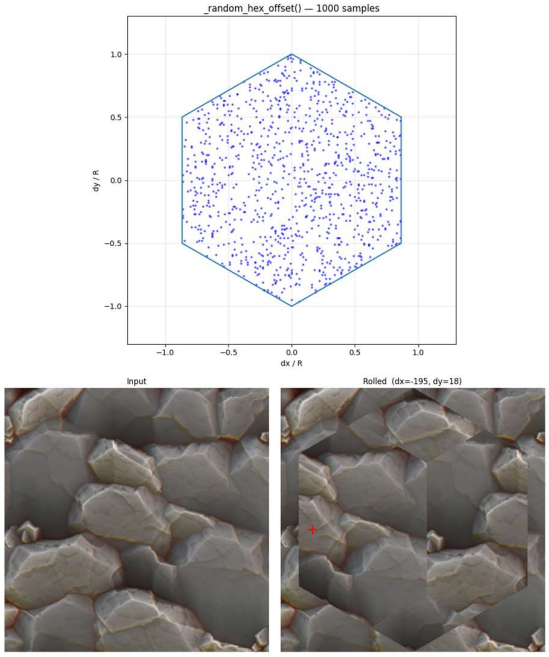


Fig. 3. Visualization of the latent manipulation modes, the random hexagonal offset, and the wrap around mechanism.

Cells outside the mask are not simply left alone between steps. At every step they are overwritten with the source texture latent brought to the current noise level, so the frozen region stays pinned to the input instead of drifting away from it. This matters more here than it would in ordinary inpainting. The periodization operator copies content outward from the hexagon interior at every step, so any drift inside the frozen region would be carried into the exterior shadow and fed straight back into the U-Net context on the following step, where it would compound. All of these mechanisms combine into a single denoising step, summarized in Algorithm 1.

Algorithm 1 One Hexagonal Denoising Step

Input: z_t, ε, z_0^{src}, m_{lat}, z_{msk}, c, step i, timestep t, latent hexagon radius R

Output: z_{t-1}

1: $z_{\text{init}} \leftarrow \Phi_R[\ \text{AddNoise}(z_0^{\text{src}}, \varepsilon, t)\ ]$
2: $z_t \leftarrow m_{\text{lat}} \odot z_t + (1 - m_{\text{lat}}) \odot z_{\text{init}}$ ▷ *re-inject frozen region*
3: $z_t \leftarrow \Phi_R[\ z_t\ ]$ ▷ *periodization, Eq. (1)*
4: $\delta \leftarrow \text{RandomHexOffset}(R)$ ▷ *(0, 0) in no-roll mode*
5: $\hat{z}, \hat{m}, \hat{z}_{\text{msk}} \leftarrow \Psi_\delta[z_t], \Psi_\delta[m_{\text{lat}}], \Psi_\delta[z_{\text{msk}}]$ ▷ *hexagonal roll*
6: $\hat{c} \leftarrow \Psi_{8\delta}[c]$ ▷ *pixel scale*
7: $\hat{z}_{\text{in}} \leftarrow \text{ScaleModelInput}(\hat{z}, t)$
8: $r_{\text{down}}, r_{\text{mid}} \leftarrow \text{ControlNet}(\hat{z}_{\text{in}}, t, \hat{c}, \lambda_{\text{cn}})$ ▷ *optional*
9: $u \leftarrow [\ \hat{z}_{\text{in}} \,\|\, \hat{m} \,\|\, \hat{z}_{\text{msk}}\ ]$ ▷ *9 channels*
10: $\hat{\varepsilon}_{\text{u}}, \hat{\varepsilon}_{\text{c}} \leftarrow \text{UNet}(u, t, e_{\text{txt}}, r_{\text{down}}, r_{\text{mid}}, e_{\text{ip}})$
11: $\hat{\varepsilon} \leftarrow \hat{\varepsilon}_{\text{u}} + g_i\,(\hat{\varepsilon}_{\text{c}} - \hat{\varepsilon}_{\text{u}})$ ▷ *CFG, g_i from schedule*
12: $\hat{z}' \leftarrow \text{SchedulerStep}(\hat{\varepsilon}, t, \hat{z})$
13: $\hat{z}' \leftarrow \Phi_R[\ \hat{z}'\ ]$ ▷ *re-periodization*
14: $z_{t-1} \leftarrow \Psi_\delta^{-1}[\ \hat{z}'\ ]$ ▷ *restore positions*

This ordering cannot be permuted. The periodization on line 3 must precede the shift, because the shift reads content

outside the hexagon as a periodic shadow. The periodization on line 13 must precede the position restore, because the scheduler only updates the hexagon interior, leaving the shadow outside it stale after one step.

The two conditioning signals enter the network at different points. ControlNet runs on the shifted four channel latent before the nine channel concatenation, and its outputs are residuals added into the U-Net down blocks and mid block. The IP Adapter embedding is attached as an additional condition and therefore enters through the cross attention layers. The U-Net input is then assembled by concatenating three parts along the channel dimension, namely four channels of noised latent normalized by the scheduler, one mask channel, and four channels of masked texture latent, for a total of nine channels as expected by the inpainting model. It must be stressed that all three are shifted by the same offset. If only the latent were shifted while the mask were not, the U-Net would receive a mask that no longer corresponds to the latent content, and regions that should have been frozen would be resynthesized instead.

Guidance uses ordinary classifier free guidance and is active only when its value exceeds one. When guidance_schedule is supplied, the guidance value at each step is computed by piecewise linear interpolation over that list against the normalized step position, so guidance strength can decay over the course of denoising. Strong guidance early helps establish structure, while weaker guidance later reduces artifacts.

Once every step is complete, regions with a zero mask are locked back to the clean source latent, so the frozen area is guaranteed to match the input exactly rather than approximately. The periodization operator is then applied once more before decoding. After VAE decoding the same operator is applied again in pixel space, this time with the pixel scale radius. This last step is necessary because the VAE decoder is convolutional with a wide receptive field, so an exactly periodic latent does not guarantee an exactly periodic pixel output near the hexagon boundary. Applying periodization in pixel space closes that gap without touching the hexagon interior, and its cost is negligible since it is a single copy operation.

E. Conditioning with IP Adapter and ControlNet

The IP Adapter [20] injects visual information from a reference texture into the U-Net through cross attention, so the generated result follows the style of that texture. Supplying a texture example directly is more appropriate than a textual description in this setting. In pass one the reference is the input texture. In pass two the reference may be swapped for the pass one output, which is generally safe because pass one introduces little new texture and its size already matches the working canvas. The risk is that a seam style is carried over when pass one still leaves a seam behind, and in such cases the original input texture is retained as the reference.

ControlNet [28] serves as an additional structural conditioning signal and is enabled only during pass two. In pass one its reference texture still contains empty regions, so the structure ControlNet sees would be wrong, while using the raw input is equally inappropriate because the structure being worked on has already changed.

F. Optional Finetuning with LoRA

As an optional module, the U-Net may first be finetuned on the input texture so that the model is slightly biased toward it. The texture is padded in reflect mode, then random crops of random size are taken and resized to the crop size so the model sees the same texture at various scales. Random masks are generated in four shapes, namely rectangles, crosses, borders, and strips, covering between ten and forty percent of the crop area, mimicking the pattern of regions that will actually be inpainted. Training uses LoRA [29] on the U-Net attention projection layers together with the input and output projections of its transformer blocks, and the training input layout is made identical to the nine channel inference layout described in Section III-D. The loss is a mixture of one half plain Mean Squared Error and one half mask weighted Mean Squared Error, so errors in the region the model is meant to fill are penalized roughly twice as heavily.

G. Inverse Mapping and Color Correction

Inverse mapping is the reverse of geometry mapping. Because the canvas size was rounded to a multiple of eight, a slight scale discrepancy exists between the original coordinates and the generated coordinates, so this scale factor is computed and used to correct all dimensions including the hexagon radius. The inpainted result is cropped to the hexagon shape with an intentional overshoot of about two pixels beyond the hexagon edge, so that no empty gaps appear from coordinate rounding when the texture is repeated, and is then tiled periodically into a rectangular texture.

The VAE decoded result may exhibit a slight color shift relative to the input. Color correction applies Reinhard color transfer [30] in the LAB color space, which rescales and re-centers each channel so that its mean and standard deviation match those of the reference. Unlike the standard application, matching is performed only on the low frequency component. Both textures are blurred with a Gaussian blur in border reflect mode, the difference against the blurred version is retained as the high frequency component, the statistics of the low frequency component are matched, and the high frequency component is then added back. This shifts the color tone toward the input without destroying the newly generated high frequency detail.

H. Implementation

The method is implemented as a modular Python library divided into geometry, diffusion, conditioning, evaluation, observation, and utility modules. As a proof of concept, the library is also wrapped in a web service with a job queue and separate worker processes, so users can upload a texture, set hyperparameters, watch logs live, and view the result mapped onto a three dimensional object. Since the scientific contribution of this paper lies in the generation method, the web service is not discussed further.

IV. Experimental Design

A. Environment and Dataset

All experiments run on a single NVIDIA GeForce RTX 3060 Laptop GPU with 6 GB of VRAM, Windows 11, PyTorch with the diffusers 0.37.0 library, the runwayml/stable-diffusion-inpainting base model, and an fp32 precision VAE.

The test dataset consists of 14 textures split into three regularity categories following SeamlessGAN practice, namely regular with 5 textures having structured repeating patterns, near stochastic with 5 semi random textures, and stochastic with 4 homogeneously random textures. This split matters because hyperparameter effects differ by texture type. For the direct comparison against prior work, the same dataset used in the SeamlessGAN evaluation is used.

B. Metrics and Measurement Procedure

The quality of a tileable texture cannot be summarized in a single number, so it is measured along two aspects. Fidelity is measured with SSIM, LPIPS, SI-FID, and DISTS, while tileability is measured with TexTile. SSIM, SI-FID, and LPIPS follow the SeamlessGAN evaluation so results remain connected to prior work, whereas DISTS and TexTile are extensions introduced here. SSIM is still reported but is not treated as the primary reference because it is highly sensitive to spatial displacement, so two textures that are identical but shifted by a few pixels can score very poorly.

Because the system output is hexagonal while the input is square, an adjustment is required. For reference based metrics, the output is cropped at the hexagon radius about the canvas center, resized to the input size, and compared against the input. For TexTile, which is reference free, the output is first tiled hexagonally into a canvas the size of the rectangular unit cell of the hexagonal lattice, so that the result can be tiled squarely as the metric expects.

C. Reference Configuration and Experiment Structure

The reference configuration that anchors all experiments is given in Table I. Finetuning and ControlNet are disabled by default because they proved unprofitable, and both are tested in the ablation study in the opposite direction, that is, enabled from the reference configuration.

TABLE I. Reference Configuration

Group	*Parameter*	*Value*
Diffusion	num_inference_steps	20
Diffusion	guidance_scale	10
Diffusion	strength	1.0
Diffusion	seed	481
Latent manipulation	roll_mode	hex_copy_no_roll
Multi-pass exterior	status	enabled
Multi-pass exterior	exterior_steps	20
Multi-pass exterior	exterior_guidance_scale	10
Multi-pass exterior	exterior_strength	1.0
Geometry	hypotenuse	256
Geometry	outer_margin	75
Geometry	inner_padding	10
Geometry	gap_padding	10
Geometry	camera_padding	0
IP Adapter	scale	1.0
Postprocess	color_postprocess_strength	1.1
Postprocess	color_postprocess_blur_sigma	512
Finetune	status	disabled
ControlNet	status	disabled

One configuration means one complete set of values for every hyperparameter, so changing a single value already yields a different configuration. The experiments are laid out as a configuration by image matrix. Each configuration is run over all 14 textures and its metrics are averaged, so each configuration produces one row of results as a mean and standard deviation. From the reference configuration, 43 configurations are built, namely 1 reference, 28 One Factor At a Time (OFAT) variants over 14 parameters, and 14 ablation and interaction configurations, for a total of 602 runs. All of them completed without a single failure.

The seed is held constant across configurations, so score differences stem purely from the configuration and the data is paired. Significance is therefore tested per texture with the Wilcoxon signed rank test [31] against the reference configuration, which separates hyperparameter effects from random variation across images. As an extreme baseline, raw hex tiling is used, namely the input texture tiled hexagonally with no diffusion at all. This baseline has perfect fidelity by construction, so it serves as both the fidelity ceiling and the tileability floor that must be exceeded.

V. Result And Discussion

A. Main Results

Table II reports the reference configuration results for each texture category.

TABLE II. Average Metrics and Duration per Texture Category

Category	*SSIM* ↑	*LPIPS* ↓	*SI-FID* ↓	*DISTS* ↓	*TexTile* ↑	*Duration (s)*
Regular	0.080 ± 0.028	0.493 ± 0.076	6.77 ± 1.81	0.237 ± 0.023	0.637 ± 0.155	33.9
Near Stochastic	0.115 ± 0.110	0.392 ± 0.134	10.35 ± 3.90	0.263 ± 0.026	0.444 ± 0.238	37.0
Stochastic	0.221 ± 0.082	0.460 ± 0.095	12.01 ± 5.41	0.249 ± 0.072	0.685 ± 0.156	49.4

Category	*SSIM* ↑	*LPIPS* ↓	*SI-FID* ↓	*DISTS* ↓	*TexTile* ↑	*Duration (s)*
Average	0.133 ± 0.095	0.447 ± 0.107	9.55 ± 4.19	0.250 ± 0.041	0.581 ± 0.206	39.5

Tileability depends heavily on texture type. Stochastic textures are easiest to tile at a TexTile of 0.685 because their randomness disguises the seam, whereas near stochastic textures are hardest at 0.444 because their half regular structure makes the seam most conspicuous. Regular textures sit between the two at 0.637.

The low SSIM of 0.133 is not an indication of failure but a property of the metric. The output is resynthesized and displaced hexagonally due to outer_margin, so a low SSIM is a natural consequence even when the visual result is good. Fidelity is better read through DISTS and LPIPS. Notably, the worst SI-FID occurs in the stochastic category at 12.01 despite it having the highest TexTile, which underlines that distributional fidelity is a different thing from tileability.

One generation typically completes in about 40 seconds. The variation in the duration column across categories, namely 33.9 to 49.4 seconds, comes from model warm up between generations rather than from any property of the categories.

B. Ablation Study

Table IV changes one component of the reference configuration per row, with the last row holding the extreme raw hex tiling baseline.

TABLE III. COMPONENT ABLATION STUDY

Category	*SSIM* ↑	*LPIPS* ↓	*SI-FID* ↓	*DISTS* ↓	*TexTile* ↑
Reference	0.133	0.447	9.55	0.250	0.581
Exterior disabled	0.139	0.450	10.67	0.259	0.593
Postprocess disabled	0.138	0.448	9.58	0.251	0.578
ControlNet disabled	0.126	0.460	11.05	0.278	0.552
Finetune enabled	0.134	0.451	9.88	0.257	0.590
Raw hex tiling	1.000	0.000	0.00	0.000	0.437

Category	*Δ TexTile*	*p*	*Time (s)*
Reference	-	-	39.5
Exterior disabled	+0.012	0.670	29.1
Postprocess disabled	−0.004	0.153	71.7
ControlNet disabled	−0.030	0.542	52.2
Finetune enabled	+0.009	0.808	597.4
Raw hex tiling	−0.144	0.030	< 1

The p values come from a paired Wilcoxon test on TexTile with n = 14. The time column was measured sequentially on a single GPU and still includes model warm up between sessions, so time comparisons across rows other than the finetune row are indicative only.

The most important finding is in the last row. Raw hex tiling has perfect fidelity by construction, yet its TexTile is only 0.437 ± 0.169. The reference configuration raises it to 0.581, an improvement of 0.144 that is significant at p = 0.030 and improves 10 of 14 textures. In other words, sacrificing part of the fidelity, namely DISTS from 0.000 to 0.250, does yield a real gain in tileability. Because the reference configuration uses hex_copy_no_roll mode, this gain comes entirely from the combination of hexagonal periodization, two pass inpainting, and IP Adapter conditioning.

Disabling the exterior pass actually raises TexTile slightly by 0.012 and only marginally worsens DISTS by 0.010, neither of which is significant at p = 0.670 and p = 0.296 respectively. The single pass configuration is also 26 percent faster. The benefit of multi pass therefore lies in the visual quality of the exterior region, which the metrics do not fully capture because they are computed on the tile region only. Postprocessing has an almost null effect on the metrics at −0.004, since its role is limited to a global color tone correction that is purely visual.

C. Hyperparameter Sensitivity

Table V summarizes the effect of selected factors on TexTile and DISTS relative to the reference configuration.

TABLE IV. SENSITIVITY OF SELECTED HYPERPARAMETERS

Factor	*Value*	*TexTile (Δ)*↑	P_T	*DISTS (Δ)*↓	P_D
outer margin	0	0.534 (−0.047)	-	0.217 (−0.033)	-
	150	0.627 (+0.045)	0.715	0.291 (+0.041)	0.003
Camera padding	50	0.625 (+0.043)	-	0.284 (+0.034)	-
	100	0.644 (+0.063)	0.104	0.270 (+0.020)	0.068
Inner padding	75	0.550 (−0.032)	0.194	0.308 (+0.058)	0.001
	150	0.531 (−0.050)	-	0.337 (+0.088)	-
Hypotenuse	206	0.564 (−0.017)	-	0.306 (+0.056)	-
	306	0.589 (+0.008)	-	0.263 (+0.013)	-
Guidance Scale	1	0.620 (+0.039)	0.194	0.251 (+0.002)	0.855
	5	0.610 (+0.029)	0.078	0.245 (−0.005)	0.542
Guidance Schedule	[7,3,1]	0.619 (+0.038)	0.078	0.244 (−0.006)	0.463
Strength	0.7	0.630 (+0.049)	0.135	0.247 (−0.003)	0.670
	0.9	0.615 (+0.034)	0.268	0.248 (−0.002)	0.903
Num inference steps	5	0.605 (+0.023)	0.217	0.276 (+0.027)	0.003
	50	0.573 (−0.008)	-	0.254 (+0.004)	-

The reference configuration scores TexTile 0.581 and DISTS 0.250. A dash marks variants that were not part of the Wilcoxon test set. The roll_mode values are derived from the paired differences against the reference.

outer_margin is the strongest control over the fidelity and tileability trade off. Consistent with its mechanism, a value of zero makes the camera cover almost nothing but the hexagon, so the output closely resembles the input and SSIM jumps to 0.529, but the room available to harmonize the seam becomes narrow. Conversely, a value of 150 widens the exterior region so TexTile rises to 0.627 at a significant cost in fidelity. camera_padding offers a more favorable pattern, raising TexTile to 0.644 at a far smaller fidelity cost, because the extra border space supplies context for inpainting without enlarging the region that must be resynthesized

On the diffusion side, a strength of one turns out to be overly aggressive, since it starts from pure noise and discards the exterior structure produced by pass one, whereas values of 0.7 to 0.9 preserve structure so that both fidelity and tileability improve. Visually, however, those values can produce faint seams that TexTile does not capture well. Lowering guidance_scale or using a decaying guidance schedule consistently raises TexTile without hurting DISTS. Adding denoising steps does not scale linearly with quality, since 50 steps is slightly worse than 20 at a much greater time cost, while 5 steps already raises TexTile even though fidelity degrades significantly.

Both modes with shifting enabled trend favorably on both aspects at once, namely a TexTile of 0.637 for hex mode and 0.622 for square mode, with DISTS improving as well. These gains do not reach significance at n = 14, but their direction is consistent and suggests that the shifting mechanism deserves to become the default configuration in a larger scale evaluation.

The secondary hyperparameters, namely exterior_steps, exterior_guidance_scale, color_postprocess_strength, color_postprocess_blur_sigma, and gap_padding, all produce small effects with TexTile changes of no more than 0.015 in magnitude. This indicates that the pipeline is insensitive to that group of parameters. The single exception is an overly small color_postprocess_blur_sigma, since the color matching then only captures small isolated areas and introduces localized color changes..

D. The Tileability and Fidelity Trade-off

The Wilcoxon tests reveal a clear pattern. Not a single variant produces a significant TexTile improvement over the reference configuration, with the smallest p value being 0.078. Fidelity degradation, by contrast, is far more decisive. Using inner_padding of 75 at p = 0.001, together with outer_margin of 150, num_inference_steps of 5, and enabling ControlNet, all at p = 0.003, demonstrably raises DISTS, meaning fidelity drops. In other words, the fidelity loss caused by parameters designed to improve tileability is a statistically supported finding, whereas tileability gains at n = 14 still fall within the range of variation across textures. The only significant tileability gain in this work is that of the reference configuration over raw hex tiling.

E. Heterogeneity Across Categories

Many factors have different, sometimes opposite, effects depending on texture type. Table VI shows the change in TexTile per category relative to each category's own reference, namely 0.637 for regular, 0.444 for near stochastic, and 0.685 for stochastic.

TABLE V. TEXTILE CHANGE PER TEXTURE CATEGORY

Factor	***Reguler***	***Near Stochastic***	***Stochasstic***	***Pattern***
guidance_scale = 1	+0.032	+0.036	+0.052	helps all
outer_margin = 150	+0.017	+0.121	−0.014	helps near stochastic
inner_padding = 75	−0.144	+0.119	−0.079	opposing
ControlNet enabled	−0.097	−0.040	+0.067	opposing
num_inference_steps = 5	+0.034	−0.031	+0.077	helps regular and stochastic
hypotenuse = 206	−0.071	+0.017	+0.009	hurts regular
Finetune enabled	+0.003	−0.003	+0.029	negligible

Lowering guidance_scale raises TexTile in all three categories, so it is safe to apply across the board. Conversely, inner_padding of 75 helps near stochastic by 0.119 but hurts regular by 0.144, and this is why its overall average is negative despite being beneficial on the hardest category. A similar pattern holds for ControlNet, which helps stochastic by 0.067 but hurts regular by 0.097. The hardest category, near stochastic, benefits most from geometry configuration, and the combination of a large outer_margin and inner_padding lifts it from 0.444 to roughly 0.56. This finding indicates that configuration selection should depend on texture category rather than being a single setting for all cases.

F. Negative Results for Finetuning and ControlNet

Finetuning barely raises TexTile. All of its changes, ranging from −0.007 to +0.022, lie well inside the TexTile standard deviation of 0.206, with improvements on only 6 of 14 textures and p = 0.808. Finetuning also consistently degrades fidelity, raising DISTS by 0.008 to 0.016, because LoRA nudges the model slightly away from the properties of the original texture. Its own hyperparameters have almost no effect either, since raising steps from 100 to 600 only moves TexTile from 0.592 to 0.603, and changing the learning rate from 1e-5 to 1e-4 only moves it from 0.574 to 0.591. At a cost of roughly nine minutes of training per texture, the reward is not commensurate..

ControlNet is harmful on both aspects, and its fidelity degradation is significant at p = 0.003. The larger the conditioning_scale, the worse the result, with TexTile at 0.567 for a value of 0.5 and 0.558 for a value of 1.0. The most plausible explanation is that the IP Adapter already serves as an adequate conditioning source, so adding ControlNet is generally

redundant and instead forces the model to preserve the structure produced by pass one, which is not necessarily seam free. The only exception is the stochastic category.

Both negative results are important to report because both are components that intuition suggests should help, yet in practice they only add cost. Disabling them is a design decision supported by data.

G. Inter Metric Correlation

A Pearson correlation matrix was computed over all 602 runs, with each run contributing one data point of five metric scores. SSIM is nearly orthogonal to the perceptual metrics, at 0.01 against LPIPS and 0.10 against DISTS, reinforcing the conclusion that SSIM measures something different and is poorly suited to hexagonal tileable textures. SI-FID and DISTS correlate strongly at 0.73, so the two are largely redundant as fidelity measures. LPIPS correlates positively with TexTile at 0.51, and since a high LPIPS means poor fidelity while a high TexTile means good tileability, this figure is direct quantitative evidence of the trade off between fidelity and tileability. By contrast, DISTS and SI-FID correlate only weakly with TexTile at −0.12 and −0.10, so both can even improve alongside TexTile. No single metric represents overall quality..

H. Computational Cost

The full experiment set consumed 23 hours and 26 minutes of pure compute time. The mean per run is 140.2 seconds with a median of 45.0 seconds and a standard deviation of 298.6 seconds, while the fastest run completed in 24.0 seconds and the slowest in 34 minutes. The gap between mean and median indicates a right skewed distribution, that is, most runs finish in about 45 seconds while a handful are very slow because of finetuning.

The breakdown confirms this. The 70 runs that use finetuning, or 11.6 percent of the total, consumed 67 percent of all compute time, averaging 808 seconds per run or roughly 15.4 times longer than an ordinary run. Disabling finetuning cuts two thirds of the total time with no loss in quality. For the reference configuration, pass one contributes about 10 seconds, each scheduler step about 0.6 seconds, and ControlNet about 13 seconds.

I. Qualitative Evaluation

Placing several tiles side by side shows that the system successfully produces smooth joins from an input that does not join smoothly, whether compared against raw hex tiling or raw square tiling. The hardest cases arise with near stochastic textures containing both random areas and a small region that tends toward regularity, so the model struggles to decide which pattern to continue across the boundary.

A visual comparison between multi pass and single pass shows that single pass produces a fairly conspicuous hexagonal join pattern, whereas multi pass yields a better output because the IP Adapter in pass two already uses a reference texture at the appropriate size. This difference is not captured by the metrics because they are computed on the tile region only, which explains why Table IV makes the exterior pass appear useless. The same holds for color correction, which is nearly metric neutral yet visually prevents the output from turning slightly darker than the input.

Fig. 4. Comparison of raw hex tiling, raw square tiling, and this method's output tiled hexagonally. The red hexagon outlines a single tile.

Fig. 5. Visual comparison of multi pass against single pass, and the effect of color correction on the output..

J. Comparison with Prior Work

Table VII compares the characteristics of the method against SeamlessGAN as the primary reference, and Table VIII gives the quantitative comparison on the same dataset used in the SeamlessGAN evaluation.

TABLE VI. METHOD CHARACTERISTIC COMPARISON

Aspect	***SeamlessGAN***	***This Method***
Model	GAN	Latent Diffusion
Per texture	required, about 40 minutes	optional, about 15 minutes
Tileability mechanism	two by two latent tiling and central crop	multi pass inpainting and hexagonal latent manipulation
Tile Shape	square	hexagonal
Quality Selection	per texture discriminator	general purpose TexTile
Evaluation Metrics	SSIM, SI-FID, LPIPS	SSIM, SI-FID, LPIPS, DISTS, TexTile
Output	texture stack of albedo and normal, double the input resolution	single RGB
Time per texture	45 minutes (GTX 1080 Ti)	24 seconds to 33.7 minutes (RTX 3060)

TABLE VII. QUANTITATIVE COMPARISON WITH PRIOR METHODS

Method	***SSIM↑***	***SI-FID ↓***	***LPIPS ↓***
Histogram Blending [5]	0.1424	1.3471	0.6207
Graph Cuts [7]	0.2086	0.9529	0.5818
Texture Stationarization [6]	0.1968	0.7620	0.5171

Method	*SSIM↑*	*SI-FID ↓*	*LPIPS ↓*
Repeated Pattern Detection [8]	0.2144	1.2958	0.5137
PSGAN [10]	0.1723	1.4355	0.5624
Self-Organizing Textures [9]	0.1753	1.3171	0.5328
SeamlessGAN [11]	0.2341	0.6311	0.4792
SeamlessHexDiffusion	0.1596	7.65	0.4851

The figures in the first six rows and for SeamlessGAN are quoted from [11]. Note that every comparison method produces square tiles whereas this method produces hexagonal tiles, so the comparison is indicative rather than strict.

The LPIPS score of this method, 0.4851, is nearly on par with SeamlessGAN and better than the other six prior methods. This means that perceptually, the hexagonal texture produced stays that close to the input even though the tile shape differs and every pixel is resynthesized.

The SSIM score is lower than SeamlessGAN's, which is consistent with the analysis in Sections V-A and V-G, namely that SSIM is highly sensitive to structural change and spatial displacement while this method's output is by design displaced hexagonally. The considerably higher SI-FID likewise cannot be read as worse seam quality. That metric tends to favor patch based methods that preserve input pixels, whereas this method resynthesizes every pixel through the VAE and the diffusion process, so its feature statistics necessarily change even when the texture stays visually consistent. SeamlessGAN itself notes that SI-FID magnitudes vary significantly between tables in its own paper, which suggests the metric is overly sensitive to global statistics [11]. The fidelity of this method is therefore better judged from LPIPS, DISTS, and visual comparison.

The clearest advantage lies in efficiency. SeamlessGAN mandates per texture adversarial training of roughly 45 minutes on a GTX 1080 Ti, whereas the reference configuration of this method completes one texture in about 40 seconds on an RTX 3060 Laptop with only 6 GB of VRAM, with no training at all.

K. Limitations

Several limitations should be noted. First, the test dataset holds only 14 textures, so its statistical power is limited, and this is why many TexTile gains fail to reach significance despite consistent direction. Second, the reference configuration was assembled using the same textures used for evaluation, which risks an optimistic bias. Third, all experiments used a single seed, so the influence of random variation remains unmeasured. Fourth, the system generates only an albedo map and does not yet handle a complete PBR material set. Fifth, the inpainting model in use requires the context region to be rectangular, so when shifting is active the model still sees a hexagonal join in its context, which limits how far the shifting mechanism can be exploited.

VI. CONCLUSION

This paper proposed SeamlessHexDiffusion, a method for generating seamless tileable hexagonal textures on top of a pretrained latent diffusion model. Hexagonal tileability is formulated as an inpainting problem by way of geometry mapping, and periodicity is then enforced through latent space manipulation consisting of hexagonal periodic copying and random shifting on the hexagonal lattice during denoising, with an IP Adapter carrying the style of the input texture. The denoising loop combines both operators with per step re-injection of the frozen region, nine channel input assembly performed after shifting, and double periodization in both latent and pixel space.

Evaluation over 602 runs shows that the system runs reliably without a single failure, and that tileability as measured by TexTile rises significantly from 0.437 for a texture tiled hexagonally without diffusion to 0.581 at $p = 0.030$. Sensitivity analysis shows that the hexagonal camera geometry, particularly outer_margin and camera_padding, is the dominant control over the fidelity and tileability trade off, and that hyperparameter effects can run in opposite directions across texture categories, so the best configuration is category specific. Two modules that intuition suggests should help, namely per texture LoRA finetuning and ControlNet, proved not to be worth their cost, and finetuning alone consumed two thirds of the total compute time without delivering a meaningful improvement. On the metric side, SSIM and SI-FID are poorly suited to this problem because they are overly sensitive to structural displacement and feature statistic changes, whereas LPIPS and DISTS are more appropriate for fidelity and TexTile proves adequate for tileability. Compared with prior work, this method attains an LPIPS nearly on par with SeamlessGAN and better than six other methods, while being far more efficient since it requires no per texture training.

Future work can proceed along several directions. Testing should be extended to hundreds of textures not involved in assembling the configuration and run with multiple seeds, so that trends that are currently consistent but not yet significant, particularly the shifting mechanism, can be tested conclusively. The approach could also be extended to generate a complete PBR material set, for instance by generating the albedo map first and then using it as a reference for the other material maps, since the hexagonal mechanism developed here does not depend on the kind of map being processed. Finally, exploring inpainting methods that can accept a non rectangular context region would directly address the main limitation of the shifting mechanism in this work.

REFERENCES

[1] S. A. Baraheem, T.-N. Le, and T. V. Nguyen, "Image synthesis: a review of methods, datasets, evaluation metrics, and future outlook," Artificial Intelligence Review, vol. 56, pp. 10813–10865, 2023.

[2] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in Proc. 28th Annu. Conf. Comput. Graph. Interact. Tech. (SIGGRAPH), 2001, pp. 341–346.

[3] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: image and video synthesis using graph cuts," ACM Trans. Graph., vol. 22, no. 3, pp. 277–286, 2003.

[4] J. Andersson, "Terrain rendering in Frostbite using procedural shader splatting," in ACM SIGGRAPH 2007 Courses, 2007, pp. 38–58.

[5] T. Deliot and E. Heitz, "Procedural stochastic textures by tiling and blending," in GPU Zen 2: Advanced Rendering Techniques, W. Engel, Ed. Black Cat Publishing, 2019.

[6] J. Moritz, S. James, T. S. F. Haines, T. Ritschel, and T. Weyrich, "Texture stationarization: turning photos into tileable textures," Computer Graphics Forum, vol. 36, no. 2, pp. 177–188, 2017.

[7] Z. Li, M. Shafiei, R. Ramamoorthi, K. Sunkavalli, and M. Chandraker, "Inverse rendering for complex indoor scenes: shape, spatially-varying lighting and SVBRDF from a single image," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 2020, pp. 2475–2484.

[8] C. Rodriguez-Pardo, S. Suja, D. Pascual, J. Lopez-Moreno, and E. Garces, "Automatic extraction and synthesis of regular repeatable patterns," Computers & Graphics, vol. 83, pp. 33–41, 2019.

[9] E. Niklasson, A. Mordvintsev, E. Randazzo, and M. Levin, "Self-organising textures," Distill, 2021, doi: 10.23915/distill.00027.003.

[10] U. Bergmann, N. Jetchev, and R. Vollgraf, "Learning texture manifolds with the periodic spatial GAN," in Proc. Int. Conf. Mach. Learn. (ICML), 2017, pp. 469–477.

[11] C. Rodriguez-Pardo and E. Garces, "SeamlessGAN: self-supervised synthesis of tileable texture maps," IEEE Trans. Vis. Comput. Graphics, 2022, doi: 10.1109/TVCG.2022.3143615.

[12] O. Madar and O. Fried, "Tiled Diffusion," arXiv:2412.15185, 2024.

[13] R. M. Mersereau, "The processing of hexagonally sampled two-dimensional signals," Proc. IEEE, vol. 67, no. 6, pp. 930–949, 1979.

[14] E. Hoogeboom, J. W. T. Peters, T. S. Cohen, and M. Welling, "HexaConv," in Proc. Int. Conf. Learn. Represent. (ICLR), 2018.

[15] J. Luo, W. Zhang, J. Su, and F. Xiang, "Hexagonal convolutional neural networks for hexagonal grids," IEEE Access, vol. 7, pp. 142738–142747, 2019.

[16] M. Mikkelsen, "Practical real-time hex-tiling," Journal of Computer Graphics Techniques, vol. 11, no. 2, pp. 77–94, 2022.

[17] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 2022, pp. 10684–10695.

[18] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," Foundations and Trends in Machine Learning, vol. 12, no. 4, pp. 307–392, 2019.

[19] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: convolutional networks for biomedical image segmentation," in Proc. Med. Image Comput. Comput.-Assist. Interv. (MICCAI), 2015, pp. 234–241.

[20] H. Ye, J. Zhang, S. Liu, X. Han, and W. Yang, "IP-Adapter: text compatible image prompt adapter for text-to-image diffusion models," arXiv:2308.06721, 2023.

[21] C. Rodriguez-Pardo, D. Casas, E. Garces, and J. Lopez-Moreno, "TexTile: a differentiable metric for texture tileability," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 2024.

[22] K. Ding, K. Ma, S. Wang, and E. P. Simoncelli, "Image quality assessment: unifying structure and texture similarity," IEEE Trans. Pattern Anal. Mach. Intell., vol. 44, no. 5, pp. 2567–2581, 2022.

[23] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," IEEE Trans. Image Process., vol. 13, no. 4, pp. 600–612, 2004.

[24] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 2018, pp. 586–595.

[25] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2017, pp. 6626–6637.

[26] T. R. Shaham, T. Dekel, and T. Michaeli, "SinGAN: learning a generative model from a single natural image," in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), 2019, pp. 4570–4580.

[27] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," in Proc. Int. Conf. Learn. Represent. (ICLR), 2021.

[28] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), 2023, pp. 3836–3847.

[29] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: low-rank adaptation of large language models," in Proc. Int. Conf. Learn. Represent. (ICLR), 2022.

[30] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley, "Color transfer between images," IEEE Computer Graphics and Applications, vol. 21, no. 5, pp. 34–41, 2001.

[31] F. Wilcoxon, "Individual comparisons by ranking methods," Biometrics Bulletin, vol. 1, no. 6, pp. 80–83, 1945.