Learning Based Infinite Terrain Generation with Level of Detailing

Aryamaan Jain[1,2], Avinash Sharma[1,3], K S Rajan[1]
[1] IIIT Hyderabad, India
[2] Inria, Université Côte d'Azur, France
[3] IIT Jodhpur, India
aryamaan.jain@inria.fr, avinashsharma@iitj.ac.in, rajan@iiit.ac.in

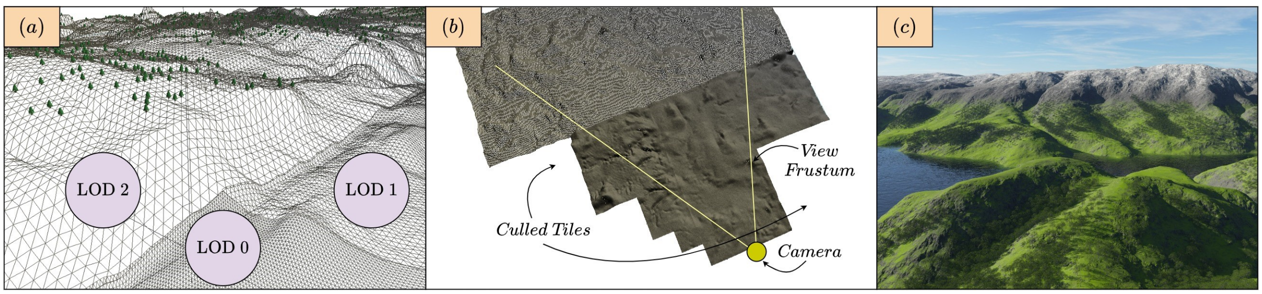

Figure 1. (a) We outline an approach for generating infinite terrain (learnt from real-world DEM), while incorporating level of detailing within a learning-based framework. (b) To efficiently manage the generated terrain data, a quad-tree structure is employed, which facilitates operations such as view frustum culling. (c) The entire process is executed in real-time, with the terrain being rendered simultaneously.

Abstract

Infinite terrain generation is an important use case for computer graphics, games and simulations. However, current techniques are often procedural which reduces their realism. We introduce a learning-based generative framework for infinite terrain generation along with a novel learning-based approach for level-of-detailing of terrains. Our framework seamlessly integrates with quad-tree-based terrain rendering algorithms. Our approach leverages image completion techniques for infinite generation and progressive super-resolution for terrain enhancement. Notably, we propose a novel quad-tree-based training method for terrain enhancement which enables seamless integration with quad-tree-based rendering algorithms while minimizing the errors along the edges of the enhanced terrain. Comparative evaluations against existing techniques demonstrate our framework's ability to generate highly realistic terrain with effective level-of-detailing.

1. Introduction

Virtual world creation is essential for modern multi-media applications such as gaming, animation or AR/VR platforms. 3D Terrain modelling and rendering are at the core of generating large-scale realistic and immersive virtual worlds (as shown in Figure 1c) as it provides the foundational structure for the environment. Terrains define the physical surface features of the Earth's crust, including mountains, valleys or plains and are popularly represented as a raster grid called Digital Elevation Models (DEM), where each cell in the grid represents a point on the surface and its elevation. Infinite terrain generation and rendering is a particular use-case finding application in games (such as *Minecraft*) or flight simulations.

Natural processes like erosion and weathering cause terrains to undergo various transformations, resulting in the development of intricate landscapes such as mountains, canyons, plateaus, and plains. Thus, a generated terrain needs to imitate these complex geometrical structures existing at multiple scales. This makes 3D terrain generation and authoring a challenging task. Existing techniques for infinite terrain generation involve procedural generation, which relies on mathematical algorithms to generate landscapes [14, 44] but fail to model these processes. The recent surge of deep learning techniques has shown promise in generating more realistic terrain, as they can learn from real-world data to produce new, varied and realistic landscapes [13, 38, 63]. However, their use for generating infinite terrain is limited, where methods like [22] leverage procedural generation along with learning which reduces their realism. We bridge this gap by proposing a fully learning-based framework for generating infinite terrain.

In the context of rendering terrains, methods [5, 9, 40] use varying levels of detail to create real-time visualizations, particularly for real-time applications. Interestingly, terrain generation and rendering were typically attempted separately. Often, terrains were generated offline, followed by online rendering or even crude online procedural terrain generation algorithms (like Perlin noise) were used, which did not approximate real-world data very well [33, 35]. Our framework can generate realistic terrain with level-of-detailing learnt from real-world data and render it simultaneously in real-time.

In this paper, we present a framework to generate and render infinite terrain with level of detail (LOD) that is learnt from real-world data using deep learning methods. Our generative module can create terrain conditioned on their neighbourhood using image completion techniques like outpainting. Our enhancement module can refine the terrain progressively based on the LOD criterion. Furthermore, we propose a novel training strategy for terrain enhancement to incorporate quad-tree-based LOD. This also enables local and global context which minimizes errors along the terrain tile edges. These modules can be seamlessly integrated with quad-tree based rendering algorithms for real-time rendering, providing a realistic and unique user experience.

In summary, we propose a generative module for infinite realistic terrain generation, an enhancement module for progressive LOD and a seamlessly integrated quad-tree based rendering algorithm that is compatible with both the generative and enhancement modules for real-time rendering. We perform an elaborate quantitative and qualitative evaluation of the proposed framework. Our code is available at https://github.com/aryamaanjain/mr_terrain.

2. Related Work

Terrain Generation involves creating a surface mesh that accurately depicts the shape and elevation of the terrain and can broadly be categorized into procedural, simulation and example/learning based methods. Procedural generation involves algorithmic generation of landscapes that exploit self-similarity or fractal structure [34] of terrain. This includes noise functions like Perlin Noise [31, 42, 44, 45], Simplex Noise [14] or subdivision schemes [8, 30, 35] like the diamond-square algorithm [10, 11]. These algorithms are popular for infinite terrain generation but do not account for the physical processes that govern the formation of terrain thereby producing unrealistic landscapes. On the other hand, simulation-based methods [2, 37] use tools to simulate the natural process of erosion over a given terrain. They are typically used to simulate particular landforms such as fluvial V-shaped valleys [3, 49] or glacial U-shaped valleys [4]. They involve modelling the flow of water and sediment across the terrain and are grounded on physical laws. They are often slow and do not lend themselves to infinite terrain generation. In contrast, learning or example based methods [17, 64] use examples or sketches of terrains to generate new terrains. Deep-learning based solutions like GANs [12, 21, 36] have been leveraged for realistic and fast terrain generation [13, 59, 63]. Furthermore, VAEs [24] have also been employed for latent space manipulation of terrains [38]. However, application of learning-based techniques for infinite terrain generation remains sparse with a recent work [22] employing a combination of diffusion models [7, 18, 39, 51] and Perlin Noise for generating infinite terrain. Although [22] does improve upon other methods, diffusion models become a bottleneck for speed while their usage of Perlin Noise for consistency along edges reduces their realism.

Image Completion [20] techniques like inpainting [43, 57] and outpainting [58] have gained attention in recent years due to the advancements in deep learning techniques and form the basis of our infinite generation framework. Inpainting involves filling in missing parts of an image. It is a challenging task as the missing parts of the image can be irregular in shape and texture. On the other hand, outpainting involves generating new content beyond the boundaries of the input image. It can be used to generate high-resolution images from low-resolution inputs, as well as to create panoramic images from a single image.

Terrain Enhancement involves adding details to the terrain. Traditionally, this has been achieved through the use of simulation erosion models [37, 54] on top of a generated terrain, which can be time-consuming. Deep learning methods have become popular for terrain enhancement, with techniques such as [13] seeking to learn erosion features to accelerate the process. Terrain enhancement can also be framed as a terrain super-resolution problem, wherein a low-resolution terrain DEM is converted to a higher resolution. Methods such as [1, 25] employ ortho-photo and depth-map pairs to super-resolve low-resolution terrain tiles using attention feedback networks. Other methods, such as [22, 26] use only the depth map. In a broader perspective, there exists work on terrain enhancement incorporating style embeddings [63] as well as leveraging GANs [6, 56]. Although there is a wide array of techniques for enhancing terrains, they all process the terrain by breaking it into smaller tiles and then enhancing them separately. This causes an accumulation of errors along the edges [26], whereas our novel quad-tree based processing minimizes this issue.

Progressive super-resolution [27, 28, 53] is a type of im-

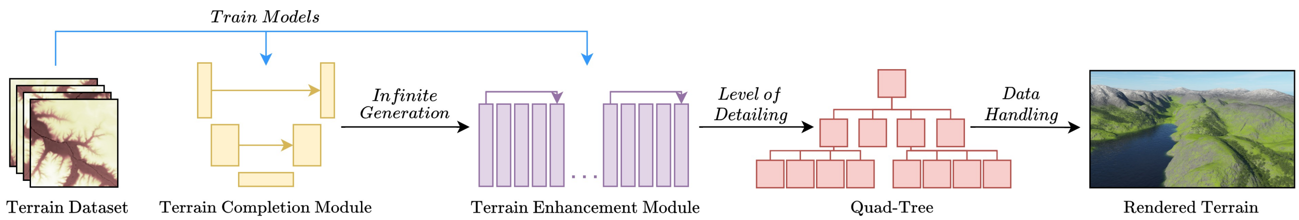

Figure 2. Overview of our framework: The terrain completion and enhancement modules are learnt from real-world datasets. The terrain completion module generates infinite terrain while the terrain enhancement module aids in level of detailing. The entire process is managed using a quad tree structure facilitating real-time rendering.

age super-resolution technique that involves creating a series of intermediate images with increasing resolution until the desired final resolution is achieved. We leverage this technique to LOD the terrain at multiple resolutions.

Terrain Rendering refers to the process of creating a visual representation of a 3D terrain. There exist many terrain rendering techniques in the literature [5, 9, 19, 32, 33, 41, 60]. Quad-tree based terrain rendering [40, 48, 55, 61] involves hierarchical decomposition of a terrain into progressively refined quadrants based on viewer proximity, enabling efficient level-of-detail management and rendering optimizations like view-frustum culling. We train our enhancement module to LOD the terrains such that it is compatible with quad-tree based rendering.

3. Method

An overview of our framework is given in Figure 2. The Terrain Completion Module (TCM) is trained to generate infinite terrain at a fixed resolution whereas the Terrain Enhancement Module (TEM) is trained to enhance the resolution of the terrain for LOD. We start with an initial fixed size DEM, which can either be taken from an existing dataset or generated via a terrain tile generation algorithm [13, 38]. We then employ the TCM to extend the initial DEM progressively based on the camera position enabling infinite generation. TCM generates terrain at the coarsest scale and needs to be enhanced for LOD based on criteria such as viewing position, direction and roughness of the terrain. The TEM is used to LOD the terrain. Finally, we use a quad-tree terrain rendering algorithm which is seamlessly integrated with the terrain completion and enhancement modules to render the terrain in real-time.

The notations used in the proceeding sections are illustrated in Figure 3. We denote the infinite terrain grid as $\mathcal{T}$. We get $s \times s$ tiles by indexing the grid as $\mathcal{T}_{i,j}$ where $i, j \in \mathbb{Z}$.

3.1. Terrain Completion Module

Initially, the grid $\mathcal{T}$ contains a single tile taken from an existing dataset or generated via a terrain tile generation algorithm. As the user moves along the grid, we populate $\mathcal{T}$ with tiles generated using the TCM within a radius $\mathcal{R}$ from the user's position. To generate an unavailable tile c, we access its available 8-neighbouring tiles n and generate c conditioned on n using TCM. After generating c, we update $\mathcal{T}$ with the new tile. We loop similarly till the end. An overview of the terrain completion module is given in Figure 4 and its inference is given in algorithm 1.

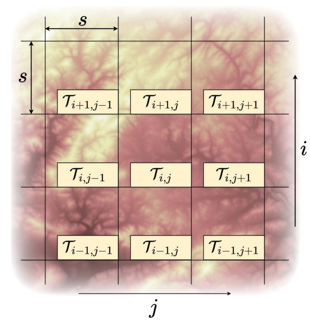

Figure 3. Illustration of the formulation and notations.

Algorithm 1: Inference of TCM

Input: x, z: Viewer position
$\mathcal{R}$: Relevance radius
$\mathcal{T}$: Terrain grid
Data: $\mathcal{G}$: Generator
forall $(u,v) \mid \|(u,v)\text{-}(x,z)\|_2 \leq \mathcal{R}$ **do**
  if *not available*$(\mathcal{T}_{u,v})$ **then**
    $n \leftarrow$ getNeigh$(u,v,\mathcal{T})$ // get available neighbour tiles
    $c \leftarrow \mathcal{G}(n)$ // predict center tile
    $\mathcal{T}_{u,v} \leftarrow c$

To train our model, we take a DEM tile t and divide it into a 3×3 grid of tiles (t_{11}, t_{12}, . . . , t_{33}). We then randomly mask the 8 boundary tiles to simulate a scenario of making them unavailable as we may encounter while running the system where $\mathcal{T}$ would be filled based on the trajectory of the user, producing arbitrary neighbourhoods. Subsequently, our objective is to predict the center of the grid t_{22} given its neighbours. More specifically, our input consists of a stack of the DEM along with 3 masks which are

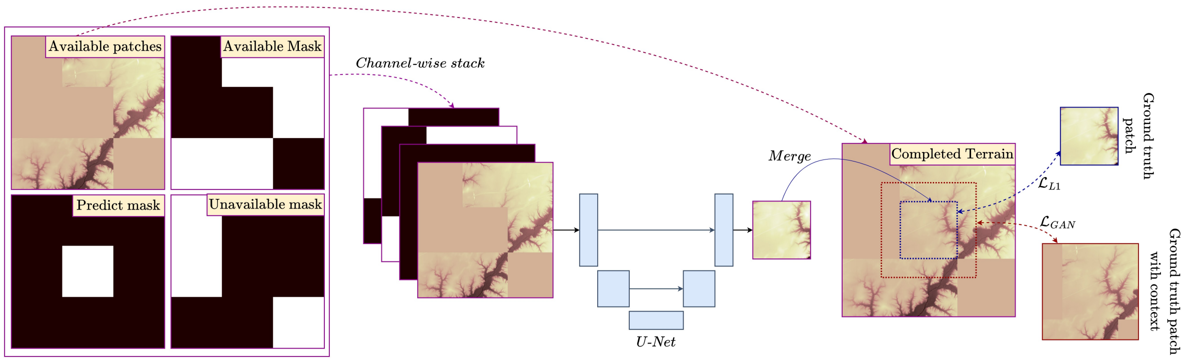

Figure 4. Overview of the terrain completion module.

1) predict mask: 1 for the t_{22} and 0 elsewhere, 2) available mask: 1 where t_{22} neighbours are available and 0 elsewhere and 3) unavailable mask: 1 where t_{22} neighbours are unavailable and 0 elsewhere.

We implement the generator $\mathcal{G}$ as a UNet [47] while the discriminator $\mathcal{D}$ is a sequential CNN followed by an MLP. We optimize for L1 and GAN losses [21]. Denoting the generated tile as $\hat{t_{22}}$, we define the $L1$ loss as,

$$\mathcal{L}_{\text{L1}}(\mathcal{G}) = \|t_{22} - \hat{t_{22}}\|_1 \tag{1}$$

Let the generated tile along with a neighbouring context extending beyond its boundary (as shown in the red dotted box in Figure 4) be denoted by $t_{22}^{\hat{c}}$ and ground truth tile along with context be denoted by t_{22}^{c}. We define the GAN loss as,

$$\mathcal{L}_{\text{GAN}}(\mathcal{G}, \mathcal{D}) = \min_{\mathcal{G}} \max_{\mathcal{D}} \log(\mathcal{D}(t_{22}^{c})) + \log(1\text{-}\mathcal{D}(t_{22}^{\hat{c}})) \tag{2}$$

Taking an extended neighbouring context provides local and global coherence [20]. The final loss for the TCM is given by,

$$\mathcal{L}_{\text{TCM}}(\mathcal{G}, \mathcal{D}) = \mathcal{L}_{\text{GAN}}(\mathcal{G}, \mathcal{D}) + \lambda_{\text{TCM}}\mathcal{L}_{\text{L1}}(\mathcal{G}) \tag{3}$$

λ_{TCM} is a hyper-parameter to adjust the contribution of the two terms.

3.2. Terrain Enhancement Module

Given a $s \times s$ tile $\mathcal{T}_{i,j}$ at a coarse resolution, we progressively enhance it for LOD. More specifically, we divide our $s \times s$ tile into four $s/2 \times s/2$ quadrant tiles and then enhance each quadrant tile to a $s \times s$ tile separately. We repeat this process recursively, with each step enhancing the resolution by a factor of 2. This is further elaborated in algorithm 2 and illustrated in Figure 5. The *shouldEnhance* function in algorithm 2 depends on the roughness of the tile, intersection with view frustum, distance from the camera and availability of enhancement models beyond a certain depth in the recursion. We cache the output of TEM in a quad-tree data structure for faster inference. This technique allows us to enhance separate parts of the terrain with varying resolutions aiding in LOD.

Algorithm 2: Recursive inference of TEM

Input: t: Tile to render
x, z: Viewer position
d: Recursion depth
Data: E_0, E_1, E_2, E_3, E_4: Enhancement models
if *shouldEnhance(t, x, z, d)* **then**
    // LOD: divide & enhance quadrants of tile recursively
    TEM($E_d(t_{\text{ne}})$, x, z, d+1)   // north-east quadrant
    TEM($E_d(t_{\text{nw}})$, x, z, d+1)   // north-west quadrant
    TEM($E_d(t_{\text{se}})$, x, z, d+1)   // south-east quadrant
    TEM($E_d(t_{\text{sw}})$, x, z, d+1)   // south-west quadrant
else
    loadBuffer(t)   // send tile to GPU
    renderTile(t)

We describe the training strategy of TEM in algorithm 3 where we employ separate models E_i for separate resolutions. Each E_i is implemented as a CNN with skip connections [15]. Given a low-resolution input tile, we divide it into its 4 quadrant tiles, randomly select one quadrant tile and enhance it with E_0. We then take the enhanced tile and repeat the same process with E_1. We iterate over this process up to a limit employing T models $E_0, \ldots, E_T$ (T=4 in our experiments) to enhance by a factor of 2^{T+1}. We use a sum of L1 losses to optimize the parameters of E_i as described in algorithm 3.

3.3. Terrain Rendering

For each frame of rendering the terrain, we start by processing the user input and updating our camera and shader parameters. We only process terrain tiles in $\mathcal{T}$ which lie within a radius $\mathcal{R}$ from the camera position. As the camera changes position, we invoke the TCM to fill in the missing terrain tiles. For each tile within a distance $\mathcal{R}$, we cull the tiles that are not in the view frustum of the camera and

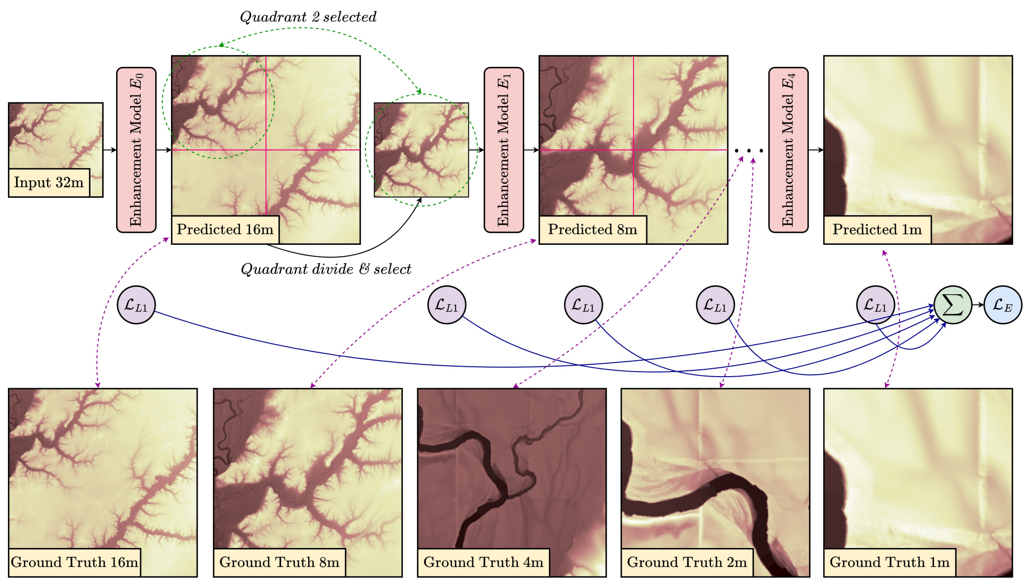

Figure 5. Overview of the terrain enhancement module.

Algorithm 3: Single step of TEM training

Data: $\hat{t}_{32}$: Low resolution input tile (32m)
$t_{16}, t_8, t_4, t_2, t_1$: High resolution ground truth tiles
E_0, E_1, E_2, E_3, E_4: Enhancement models

idx ← (32,16,8,4,2,1) // will aid in indexing
$l \leftarrow 0.0$ // loss
for $i \leftarrow 0$ **to** 4 **do**
  $q \leftarrow$ selectQuad($\hat{t}_{\text{idx}[i]}$) // select random quadrant of tile
  $q' \leftarrow$ features(q) // add features to the quadrant
  $\hat{t}_{\text{idx}[i+1]} \leftarrow E_i(q')$ // forward pass
  $l \leftarrow l$ + loss($\hat{t}_{\text{idx}[i+1]}, t_{\text{idx}[i+1]}$) // accumulate loss
Adam(∇l) // backprop & step

Algorithm 4: Single frame render step

Input: camera
TCM, TEM
$\mathcal{R}$: Relevance radius
$\mathcal{T}$: Terrain grid

ip ← userInput() // position, speed, *etc.*
updateCamera(camera, ip)
updateShaders(camera)
$x, z \leftarrow$ camera$_x$, camera$_z$
TCM($x, z, \mathcal{R}, \mathcal{T}$) // algorithm 1: fill unavailable tiles
forall $(u,v) \mid \|(u,v)-(x,z)\|_2 \leq \mathcal{R}$ **do**
  t ← $\mathcal{T}_{u,v}$ // extract tile to render
  TEM(t, x, z, 0) // algorithm 2: LOD & render

render the rest with LOD using the TEM. This process is described in algorithm 4.

Quad-tree based rendering makes optimizations like view-frustum culling very efficient (illustrated in Figure 1b), as we can discard the tiles that are not visible to the user for rendering via an inexpensive frustum-tile intersection check. Furthermore, the discrete nature of LOD in quad-tree based rendering facilitates seamless integration with super-resolution algorithms (Figure 1a), which would be less straightforward to accomplish using alternative terrain rendering algorithms.

4. Experiments and Results

Setup: We implemented our system in Python and used OpenGL/GLSL for rendering. Our models were trained using PyTorch on an NVIDIA GeForce RTX 2080 Ti GPU. To evaluate the real-time performance, we conducted experiments using a system with an NVIDIA GeForce RTX 3050 Laptop GPU, 12th Gen Intel Core i5 CPU (2.50 GHz) and 8GB RAM.

Dataset: The experiments employed the USGS National Map 3DEP Downloadable Data Collection [52] dataset. This publicly accessible dataset comprises 10000^2 DEM tiles at a 1-meter resolution. We downloaded approximately 700 GB of data, which was then subjected to a data cleaning process resulting in a dataset size reduction to approximately 600 GB. The dataset was cropped to the nearest power of two, yielding a final dataset size of 8192^2 at 1-meter resolution.

4.1. Implementation Details

Terrain Completion Module: The generator of TCM is a UNet with a bottleneck of 1024 channels. The encoder layers consist of [4, 64, 128, 256, 512, 1024] channels, while the decoder layers contain [1024, 512, 256, 128, 64, 1] channels. To enhance training stability, batch normalization was incorporated into the standard UNet architecture. The discriminator of TCM comprises a sequential CNN followed by an MLP. The discriminator consists of a head, a body, a tail and a classifier. The head converts the 4-channel input to 64 channels. The body maintains 64 channels across a series of 3 Conv-BatchNorm-LeakyReLU-Conv-Maxpool layers. Finally, the tail reduces the number of channels to 1 and passes flattened features to a two-layer MLP. We employed the Adam optimizer [23] with a learning rate (LR) of 0.0002 for the generator and 0.0001 for the discriminator, setting the parameters β_1 and β_2 to 0.5 and 0.999, respectively. An LR scheduler was employed to reduce the LR by 25% every four epochs. We trained the model using a batch size of 8 for 256 epochs. We set the λ_{TCM} value to 10.0 in Equation 3.

Terrain Enhancement Module: TEM is composed of a set of 5 CNN models for enhancing 2m, 4m, 8m, 16m and 32m DEMs. Each model contains 47497 trainable parameters to ensure real-time performance during inference. A single model in the TEM is comprised of three main components: the head, body, and tail, similar to the TCM. The head component is a convolutional layer that takes a 4-channel input, consisting of a DEM to enhance and 3 additional features and outputs 32 channels. The three additional channels in the input include the gradient along the x-axis, the gradient along the z-axis and the L1 norm of the gradient. The body component is composed of 4 Conv-ReLU layers, which maintain 32 channels. The tail component includes a Pixel-Shuffle layer [50] to increase the resolution by a factor of two, followed by a convolutional layer. A skip connection connects the head and tail components and the output is a residual that is added to the bicubic-upsampled input. The model is trained for 128 epochs with a batch size of 32, utilizing the Adam optimizer with an LR of 0.001 and a scheduler that reduces the LR by 20% every two epochs.

4.2. Terrain Completion

Comparative Assessment: We adopt the Fréchet Inception Distance (FID) metric [16] to quantitatively evaluate the TCM and present the corresponding results in Table 1. Our analysis involves a comparison with methods generating infinite terrains, encompassing Procedural and Kernel Blending approaches. Procedural algorithms such as Perlin, Simplex, and Ridge Noise functions lack adherence to geological laws or learning from real-world DEMs, resulting in comparatively higher FID. Furthermore, we illustrate Perlin and Ridge Noise terrain on the top row of Figure 6 where we observe their lack of distinctive features such as realistic valleys and ridges. Kernel Blending [22] represents a hybridization of learning-based and procedural techniques. While performing better than Procedural methods, their effectiveness is constrained due to their reliance on procedural techniques to ensure continuity along tile edges. As depicted in Figure 6, Kernel Blending approaches demonstrate a degree of realism towards the center but suffer from procedural technique influence towards the edges. TCM with 31 million parameters achieves significantly lower FID scores, being fully learning-based. An important insight is that, unlike other techniques where tile edges are approximately at mean elevation, such a constraint does not hold for the TCM as seen in Figure 6. We do not compare against simulation-based methods since they do not lend themselves to infinite generation.

Ablation Study: We conduct an ablation study to justify the choice of our parameters, as shown in Table 1. Our findings reveal a decreasing trend in FID as the ratio of generator to discriminator parameters increases. The number of parameters of the generator is denoted in the subscript of TCM. We increase the size of the generator by adjusting the bottleneck channels in the U-Net architecture, while the size of the discriminator is varied through the channel

Method	*FID*$^{\downarrow}$	$\mathcal{G}_{\text{width}}$	$\#\mathcal{G}/\#\mathcal{D}$
Perlin Noise [45]	489.79	-	-
Simplex Noise [14]	529.171	-	-
Ridge Noise	530.373	-	-
Kernel Blend [22]	274.749	-	-
$\text{TCM}_{2\text{M}}$	386.024	256	0.388
$\text{TCM}_{2\text{M}}$	215.854	256	1.536
$\text{TCM}_{8\text{M}}$	148.892	512	31.334
$\text{TCM}_{31\text{M}}$	114.376	1024	126.294

Table 1. Quantitative evaluation of infinite terrain generation techniques using the FID$^{\downarrow}$ [16] metric. The FID increases with the ratio of the number of parameters of the Generator $\mathcal{G}$ and Discriminator $\mathcal{D}$. The number of channels in $\mathcal{G}$'s bottleneck (U-Net) is represented as $\mathcal{G}_{\text{width}}$.

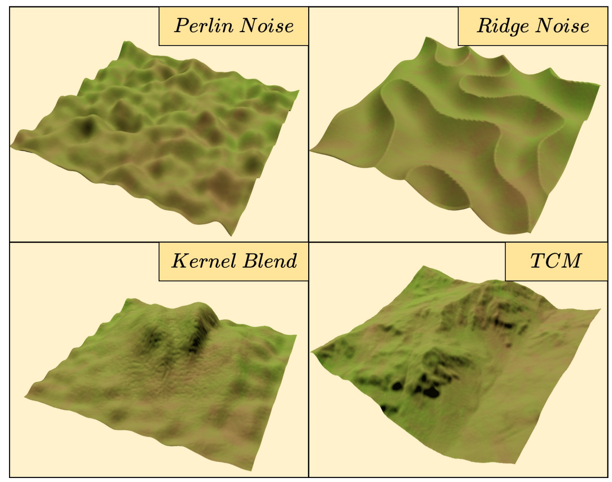

Figure 6. Qualitative comparison of terrain generation.

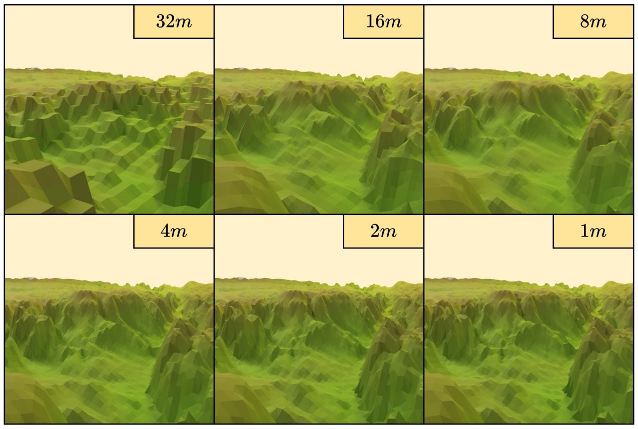

Figure 7. A 32m resolution terrain progressively enhanced with the TEM.

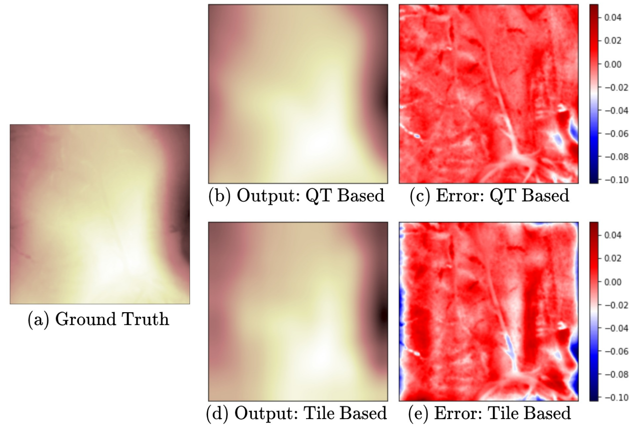

Figure 8. Comparing the residual error of quad-tree based (ours) and a tile based [22] enhancement algorithm. We observe that tile-based enhancement techniques result in a significant accumulation of errors along the tile edges.

features and body size. We cap the maximum size of the generator to 31 million parameters as networks beyond this size inhibit real-time inference.

User Study: For subjective evaluation, we conducted a user study involving 16 participants with varying levels of experience in terrain analysis. To evaluate the effectiveness of our approach, we employed a first-preference experiment methodology. Each participant was presented with 5 pairs of terrain tiles, where each pair consisted of a terrain generated using Perlin noise and TCM, randomly ordered. The participants expressed a clear preference for the terrain tile generated with TCM in 93.75% of the cases, highlighting the superior performance of our method. We have included a more detailed discussion of the user study in the supplementary material.

4.3. Terrain Enhancement

Comparative Assessment: We present a comparison of terrain enhancement based on RMSE and PSNR metrics in Table 2. AFND [25] and TRCAN [22] are tile-based super-resolution methods, *i.e.*, they enhance the entire tile for a particular factor at once, whereas our quad-tree based method progressively enhances the quads of a terrain up to a factor. AFND and TRCAN were originally designed for enhancement factors of 8, but we have trained them for factors up to 32. Our experimental results demonstrate that the TEM outperforms AFND and TRCAN for enhancement factors greater than 2 highlighting the effectiveness of the TEM for high enhancement factors.

Ablation Study: We report the results on ablation tests for TEM in Table 2. Our final model is denoted as $\text{TEM}_{47\text{K}}$. In TEM_{dem}, we pass the DEM as input without any features, which leads to lower scores. In TEM_{bprop}, we back-propagate with only the last model's loss without any direct supervision for other models. Specifically, we use the loss term $\|\hat{t}_1 - t_1\|_1$, where $\hat{t}_1$ and t_1 are the predicted tile and ground truth tiles at 1m, instead of the summation of L1 losses for all resolutions as given in algorithm 3. Despite the gradient flow through all the models, we observe poor performance for them. Interestingly, we not only observe a performance drop for factors lower than $\times 32$ but also find that the performance is not the best for $\times 32$ enhancement factor. These observations further support the potential of a progressive architecture for terrain enhancement. We also experiment with a scaled-up version of TEM, denoted as $\text{TEM}_{224\text{K}}$, which contains 224081 trainable parameters ($\times$ 5 models) compared to 47497 trainable parameters ($\times$ 5 models) of $\text{TEM}_{47\text{K}}$. While $\text{TEM}_{224\text{K}}$ scores better on RMSE/PSNR metrics, we note the trade-off between inference time and quality, which limits the practical use of a larger model. We illustrate a sample inference using the TEM in Figure 7 where finer details are added progressively, with the finest details in the 1m terrain.

Tile Edge Errors: Tile-based methods [22, 25] are known to exhibit high errors along tile boundaries, especially for high enhancement factors such as $\times 32$. This leads to a degradation in metrics like RMSE/PSNR. As illustrated in Figure 8, our proposed quad-tree based processing manages to subdue this effect. We enhance a 32m DEM to 1m using tiles-based and quad-tree based methods, compute their residual error with respect to the ground truth and plot them. We can observe particularly higher errors along the edges for the tile-based method.

Memory Constraints: One key difference between the methods prevalent in the literature of natural image super-resolution and the problem we are tackling is scale. While super-resolution on natural images has mainly been attempted for factors of up to $\times 8$ [29, 62], for terrains as in our case, the factor can be as high as $\times 32$. This makes

Method	×2 *(16m)*		×4 *(8m)*		×8 *(4m)*		×16 *(2m)*		×32 *(1m)*	
	RMSE$^{\downarrow}$	*PSNR*$^{\uparrow}$	*RMSE*$^{\downarrow}$	*PSNR*$^{\uparrow}$	*RMSE*$^{\downarrow}$	*PSNR*$^{\uparrow}$	*RMSE*$^{\downarrow}$	*PSNR*$^{\uparrow}$	*RMSE*$^{\downarrow}$	*PSNR*$^{\uparrow}$
Bicubic	0.587	39.225	0.545	39.801	0.553	39.791	0.57	39.919	0.611	39.924
AFND [25]	0.451	40.783	0.459	40.662	0.489	40.438	0.505	40.314	0.584	40.08
TRCAN [22]	0.368	41.937	0.408	41.124	0.436	40.869	0.471	40.532	0.488	40.414
TEM_{dem}	0.463	40.37	0.423	41.045	0.413	41.513	0.402	42.099	0.386	42.861
TEM_{bprop}	1.489	27.774	0.862	33.006	0.743	34.384	5.36	16.543	0.37	43.164
$\text{TEM}_{47\text{K}}$	0.417	40.817	0.395	41.495	0.384	41.974	0.374	42.557	0.359	43.294
$\text{TEM}_{224\text{K}}$	0.414	40.884	0.394	41.548	0.381	42.086	0.371	42.671	0.357	43.442

Table 2. Quantitative evaluation of terrain enhancement using the RMSE$^{\downarrow}$ (in meter) and PSNR$^{\uparrow}$ (in dB) metrics for upsampling factors from ×2 to ×32.

the direct application of methods from natural image super-resolution literature to terrains non-trivial. For example, as in our case, the input DEM is a 256^2 raster grid. Directly super-resolving it by a factor of 32 will give an output of dimensions 8192^2, which in most cases would be intractable for training due to the memory constraints of the GPU VRAM. Our proposed method is similar to progressive super-resolution, but rather than processing the entire tile, we process quadrants of the tile. This ensures that the enhancement module receives input of the same dimension at each stage unlike [27, 28, 53], where the dimensions of the input increases at successive stages limiting their scalability for higher enhancement factors due to memory constraints. Our formulation has the added advantage of being compatible with the quad-tree-based terrain rendering algorithms which have been researched quite extensively.

4.4. Rendering Details

Our framework uses OpenGL/GLSL for rendering. Our system is written in Python, enabling seamless integration with PyTorch. In Figure 1a, the LOD is showcased through a mesh representation. To optimize performance, we employ quad-trees for view frustum culling, as depicted in Figure 1b from an overhead perspective. The normals are calculated efficiently using gradients rather than cross-products and objects like trees are added using geometric instancing. We showcase a rendered scene generated and processed within our framework using Terragen [46] in Figure 1c.

In the absence of batching, TCM computes 1 inference in 0.032s on the GPU and 0.344s on the CPU. Similarly, TEM computes 1 inference in 0.0019s on the GPU and 0.017s on the CPU. In addition, we profiled the system performance by traversing a random path and recording the frames per second (FPS) at different points as shown in Figure 9. We observed occasional drops in FPS below 100 at specific points along the path, as indicated by red circles, corresponding to TCM and TEM inferences.

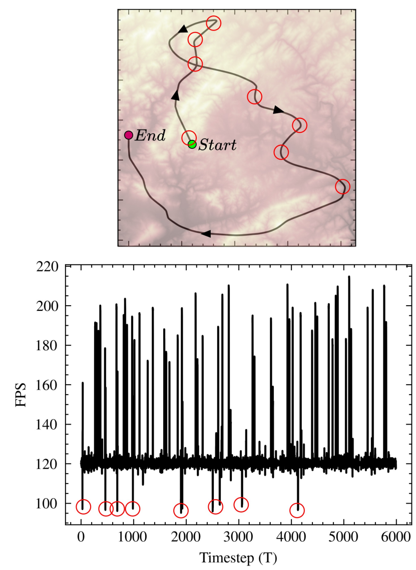

Figure 9. FPS (bottom) observed along a randomly traversed path (top). The red circles indicate the FPS dropping below 100.

5. Conclusion

We introduced a framework for learning-based infinite terrain generation and level-of-detailing compatible seamlessly with quad-tree based terrain rendering. The generative module enables the creation of vast and diverse landscapes, while the enhancement module ensures that the terrain is rendered at the appropriate level of detail. The seamless rendering algorithm provides a visually appealing and immersive experience for users. Overall, our framework offers a powerful toolset for game developers, virtual reality enthusiasts, and other applications that require high-quality large-scale terrains. Furthermore, the quad-tree based enhancement module may be extended to other domains where the raster sizes are typically large, such as the medical domain due to our memory-efficient processing. Another possible direction of exploration is the integration of other rendering algorithms or enhancing the framework's control capabilities.

References

[1] Oscar Argudo, Antoni Chica, and Carlos Andujar. Terrain super-resolution through aerial imagery and fully convolutional networks. In *Computer Graphics Forum*, pages 101–110. Wiley Online Library, 2018. 2

[2] Norishige Chiba, Kazunobu Muraoka, and Kunihiko Fujita. An erosion model based on velocity fields for the visual simulation of mountain scenery. *The Journal of Visualization and Computer Animation*, 9(4):185–194, 1998. 2

[3] Guillaume Cordonnier, Eric Galin, James Gain, Bedrich Benes, Eric Guérin, Adrien Peytavie, and Marie-Paule Cani. Authoring landscapes by combining ecosystem and terrain erosion simulation. *ACM Transactions on Graphics (TOG)*, 36(4):1–12, 2017. 2

[4] Guillaume Cordonnier, Guillaume Jouvet, Adrien Peytavie, Jean Braun, Marie-Paule Cani, Bedrich Benes, Eric Galin, Eric Guérin, and James Gain. Forming terrains by glacial erosion. *ACM Transactions on Graphics*, 42(4), 2023. 2

[5] Willem H De Boer. Fast terrain rendering using geometrical mipmapping. *Unpublished paper, available at http://www. flipcode. com/articles/article geomipmaps. pdf*, 2000. 2, 3

[6] Bekir Z Demiray, Muhammed Sit, and Ibrahim Demir. D-srgan: Dem super-resolution with generative adversarial networks. *SN Computer Science*, 2:1–11, 2021. 2

[7] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021. 2

[8] AR Dixon, GH Kirby, and Derek PM Wills. A data structure for artificial terrain generation. In *Computer Graphics Forum*, pages 37–48. Wiley Online Library, 1994. 2

[9] Mark Duchaineau, Murray Wolinsky, David E Sigeti, Mark C Miller, Charles Aldrich, and Mark B Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. In *Proceedings. Visualization'97 (Cat. No. 97CB36155)*, pages 81–88. IEEE, 1997. 2, 3

[10] Deng Fang. The study of terrain simulation based on fractal. *WSEAS Transactions on Computers*, 8(1):133–142, 2009. 2

[11] Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, 1982. 2

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 2

[13] Éric Guérin, Julie Digne, Eric Galin, Adrien Peytavie, Christian Wolf, Bedrich Benes, and Benoît Martinez. Interactive example-based terrain authoring with conditional generative adversarial networks. *Acm Transactions on Graphics (TOG)*, 36(6):1–13, 2017. 2, 3

[14] Stefan Gustavson. Simplex noise demystified. *Linköping University, Linköping, Sweden, Research Report*, 2005. 1, 2, 6

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4

[16] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017. 6

[17] Houssam Hnaidi, Eric Guérin, Samir Akkouche, Adrien Peytavie, and Eric Galin. Feature based terrain generation using diffusion equation. In *Computer Graphics Forum*, pages 2179–2186. Wiley Online Library, 2010. 2

[18] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 2

[19] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings Visualization'98 (Cat. No. 98CB36276)*, pages 35–42. IEEE, 1998. 3

[20] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and locally consistent image completion. *ACM Transactions on Graphics (ToG)*, 36(4):1–14, 2017. 2, 4

[21] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017. 2, 4

[22] Aryamaan Jain, Avinash Sharma, and K S Rajan. *Adaptive & Multi-Resolution Procedural Infinite Terrain Generation with Diffusion Models and Perlin Noise*. Association for Computing Machinery, New York, NY, USA, 2022. 2, 6, 7, 8

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[24] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 2

[25] Ashish Kubade, Diptiben Patel, Avinash Sharma, and KS Rajan. Afn: Attentional feedback network based 3d terrain super-resolution. In *Proceedings of the Asian Conference on Computer Vision*, 2020. 2, 7, 8

[26] Ashish A Kubade, Avinash Sharma, and KS Rajan. Feedback neural network based super-resolution of dem for generating high fidelity features. In *IGARSS 2020-2020 IEEE International Geoscience and Remote Sensing Symposium*, pages 1671–1674. IEEE, 2020. 2

[27] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 624–632, 2017. 2, 8

[28] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Fast and accurate image super-resolution with deep laplacian pyramid networks. *IEEE transactions on pattern analysis and machine intelligence*, 41(11):2599–2613, 2018. 2, 8

[29] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on*

computer vision and pattern recognition, pages 4681–4690, 2017. 7

[30] John P Lewis. Generalized stochastic subdivision. *ACM Transactions on Graphics (TOG)*, 6(3):167–190, 1987. 2

[31] Huailiang Li, Xianguo Tuo, Yao Liu, and Xin Jiang. A parallel algorithm using perlin noise superposition method for terrain generation based on cuda architecture. In *International Conference on Materials Engineering and Information Technology Applications (MEITA 2015)*, pages 967–974. Atlantis Press, 2015. 2

[32] Peter Lindstrom, David Koller, William Ribarsky, Larry F Hodges, Nick Faust, and Gregory A Turner. Real-time, continuous level of detail rendering of height fields. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 109–118, 1996. 3

[33] Frank Losasso and Hugues Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. In *ACM Siggraph 2004 Papers*, pages 769–776. 2004. 2, 3

[34] Benoit B Mandelbrot and John W Van Ness. Fractional brownian motions, fractional noises and applications. *SIAM review*, 10(4):422–437, 1968. 2

[35] Gavin SP Miller. The definition and rendering of terrain maps. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 39–48, 1986. 2

[36] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014. 2

[37] F Kenton Musgrave, Craig E Kolb, and Robert S Mace. The synthesis and rendering of eroded fractal terrains. *ACM Siggraph Computer Graphics*, 23(3):41–50, 1989. 2

[38] Shanthika Naik, Aryamaan Jain, Avinash Sharma, and K S Rajan. Deep generative framework for interactive 3d terrain authoring and manipulation. In *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*, pages 6410–6413, 2022. 2, 3

[39] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021. 2

[40] Renato Pajarola. Overview of quadtree-based terrain triangulation and visualization. 2002. 2, 3

[41] Renato Pajarola and Enrico Gobbetti. Survey of semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer*, 23:583–605, 2007. 3

[42] Ian Parberry. Designer worlds: Procedural generation of infinite terrain from real-world elevation data. *Journal of Computer Graphics Techniques*, 3(1), 2014. 2

[43] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016. 2

[44] Ken Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985. 1, 2

[45] Ken Perlin. Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, 2002. 2, 6

[46] Planetside. Terragen 4, 2023. 8

[47] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015. 4

[48] Stefan Röttger, Wolfgang Heidrich, Philipp Slusallek, and Hans-Peter Seidel. Real-time generation of continuous levels of detail for height fields. 1998. 3

[49] Hugo Schott, Axel Paris, Lucie Fournier, Eric Guérin, and Eric Galin. Large-scale terrain authoring through interactive erosion simulation. *ACM Transactions on Graphics*, 2023. 2

[50] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016. 6

[51] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015. 2

[52] USGS. Usgs national map 3dep downloadable data collection. `https://www.sciencebase.gov/catalog/item/543e6b86e4b0fd76af69cf4c`, 2023. [Accessed 01-Jan-2023]. 5

[53] Yifan Wang, Federico Perazzi, Brian McWilliams, Alexander Sorkine-Hornung, Olga Sorkine-Hornung, and Christopher Schroers. A fully progressive approach to single-image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 864–873, 2018. 2, 8

[54] Christopher Wojtan, Mark Carlson, Peter J Mucha, and Greg Turk. Animating corrosion and erosion. In *NPH*, pages 15–22, 2007. 2

[55] Jian Wu, Yuanfeng Yang, Sheng-rong Gong, and Zhiming Cui. A new quadtree-based terrain lod algorithm. *J. Softw.*, 5(7):769–776, 2010. 3

[56] Zherong Wu, Zhuoyi Zhao, Peifeng Ma, and Bo Huang. Real-world dem super-resolution based on generative adversarial networks for improving insar topographic phase simulation. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:8373–8385, 2021. 2

[57] Hanyu Xiang, Qin Zou, Muhammad Ali Nawaz, Xianfeng Huang, Fan Zhang, and Hongkai Yu. Deep learning for image inpainting: A survey. *Pattern Recognition*, 134:109046, 2023. 2

[58] Qingguo Xiao, Guangyao Li, and Qiaochuan Chen. Image outpainting: Hallucinating beyond the image. *IEEE Access*, 8:173576–173583, 2020. 2

[59] Jian Zhang, Chen Li, Peichi Zhou, Changbo Wang, Gaoqi He, and Hong Qin. Authoring multi-style terrain with global-to-local control. *Graphical Models*, 119:101122, 2022. 2

[60] Liwei Zhang, Jiangfeng She, Junzhong Tan, Biao Wang, and Yuchang Sun. A multilevel terrain rendering method based

on dynamic stitching strips. *ISPRS International Journal of Geo-Information*, 8(6):255, 2019. 3

[61] Lei Zhang, Ping Wang, Chengyi Huang, Bo Ai, and Wenjun Feng. A method of optimizing terrain rendering using digital terrain analysis. *ISPRS International Journal of Geo-Information*, 10(10):666, 2021. 3

[62] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu. Image super-resolution using very deep residual channel attention networks. In *ECCV*, 2018. 7

[63] Yiwei Zhao, Han Liu, Igor Borovikov, Ahmad Beirami, Maziar Sanjabi, and Kazi Zaman. Multi-theme generative adversarial terrain amplification. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019. 2

[64] Howard Zhou, Jie Sun, Greg Turk, and James M Rehg. Terrain synthesis from digital elevation models. *IEEE transactions on visualization and computer graphics*, 13(4):834–848, 2007. 2