





# Single-Shot Example Terrain Sketching by Graph Neural Networks

Y. Liu  and B. Benes 

Purdue University, West Lafayette, USA  
{liu3154, bbenes}@purdue.edu

---

## Abstract

*Terrain generation is a challenging problem. Procedural modelling methods lack control, while machine learning methods often need large training datasets and struggle to preserve the topology information. We propose a method that generates a new terrain from a single image for training and a simple user sketch. Our single-shot method preserves the sketch topology while generating diversified results. Our method is based on a graph neural network (GNN) and builds a detailed relation among the sketch-extracted features, that is, ridges and valleys and their neighbouring area. By disentangling the influence from different sketches, our model generates visually realistic terrains following the user sketch while preserving the features from the real terrains. Experiments are conducted to show both qualitative and quantitative comparisons. The structural similarity index measure of our generated and real terrains is around 0.8 on average.*

**Keywords:** geometric modellings, modelling, natural phenomena

**CCS Concepts:** • Computing methodologies → Shape modelling; Machine learning algorithms

---

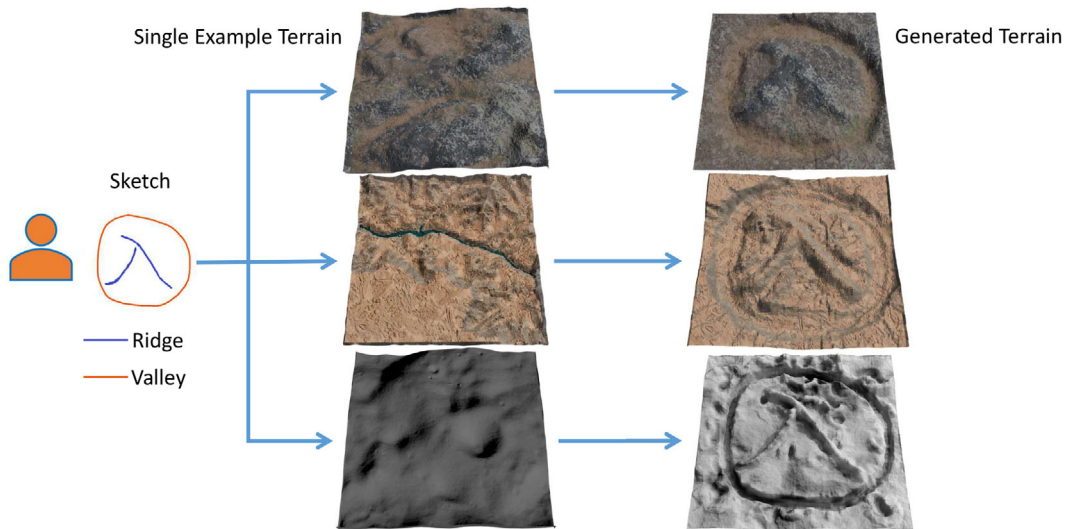
## 1. Introduction

Despite several decades of research, digital terrain authoring is still an open problem. Procedural methods provide a wide variety of terrains, but they are difficult to control and lack perceptually plausible features. Manual modelling is slow and tedious but provides reasonable control over the final shape. One interesting area includes example-based methods [GDG\*17, ZSTR07], which take real terrains and reuse them in new settings.

Thanks to the proliferation of remote-sensing sensors, the vast accessibility of digital elevation models (DEMs) gave rise to example-based algorithms and data-driven methods that can learn from the data and generate plausible digital terrain similar to the training data. Recently, Argudo et al. [AGP\*19, AAC\*17] proposed statistical learning methods to learn and generate the new terrain. Still, the statistical learning methods have worse expression power than the deep learning-based methods. Deep learning-based works like [RKČ\*22, ZLB\*19, GPM\*22, GDG\*17] applied conditional GAN [HAYC20] as a data-driven generative model, and Hu et al. [HHM\*23] applied the diffusion model for terrain generation. These methods are based on convolutional neural networks, which do not capture the topology information of a terrain. The key ingredient of these methods is the user input, and they show that their model can generate new terrain based on user-designed sketches. The sketches usually have a

different distribution than the DEM contours in the real-world training dataset. Still, the training and testing data should come from the same distribution to capture the terrain's structural properties. Also, previous works require large training data, which is not always available and does not allow a simple style definition. However, human observers can quickly tell the difference given one image from different areas, for example, the Alps, Grand Canyon, and Moon. Some works [ZLB\*19, RKČ\*22] provide 'style' transfer for terrain, but a rigorous definition of terrain style is unclear, and they use terrain data from similar places to train the model. However, these areas often include many different geological features.

We propose a sketch-based single-shot learning model based on graph neural networks (GNNs). Our model exploits the observation that the terrains resulting from the user sketch should follow the feature distribution of the example terrain input. Given a single DEM of terrain, we extract its ridges and valleys as the features, which we call a DEM contour. The assumption is that a contour will significantly affect its neighbour area and vice versa [KDRB21]. We define two processes that affect the contour neighbourhood: gather and the opposite process, scatter. The gather relates the terrain features to the contour, and the scatter distributes them. First, we take the input example terrain and extract its features. The training phase learns to reconstruct the terrain from its features by repeatedly performing the scatter and gather processes. A single shot is enough



**Figure 1:** The user selects a single image used as the main style of the output and provides an input sketch that defines the features in the generated terrain. Our method generates a new terrain that follows the input style and generates the input features. The three styles used in this image are the Alps (top), Grand Canyon (middle) and Moon surface (bottom).

for our model because it captures detailed topological and geometrical information about the features and their neighbouring area. Our model takes a user sketch during inference and generates the terrain with the target style.

An example in Figure 1 shows our approach. Three neural models were trained on the Alps, Grand Canyon, and the Moon DEMs. A user-defined sketch was used to generate new terrain models capturing the features. The model's training takes around 3 h, and the inference time is around 300 s for a DEM of a resolution  $512 \times 512$ . We claim the following contributions:

1. An end-to-end neural network-based model for single-shot terrain generation capable of learning from one single terrain DEM and generating similar terrains according to both user-defined sketches and contours from the real-world scenario.
2. The novel deep neural model simultaneously learns the topology and geometry information and is designed to ensure rotational invariance.
3. The neural network model disentangle the ridges from valleys, which is applicable to unseen cases or cases from other distributions.

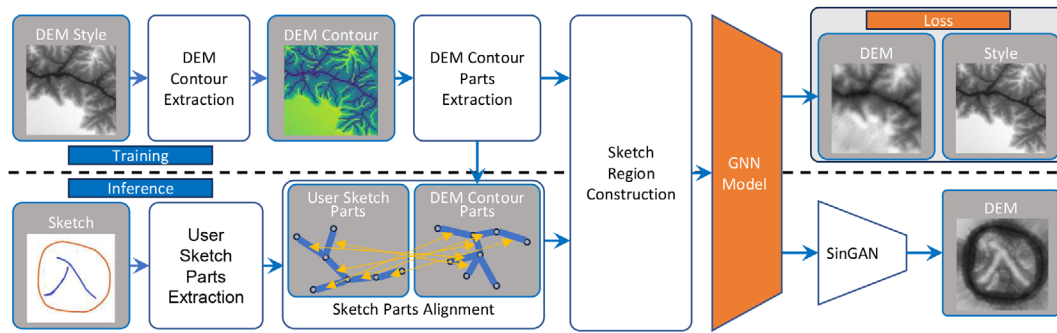
## 2. Related Work

**Terrain modelling** (see the review [GGP\*19]) has been an active area in Computer Graphics since its inception. Early works modelled terrains as fractals [FFC98, Man83], but it was quickly noticed that the scaling self-similarity does not fully capture geomorphological processes in Nature. Follow-up works then modelled various aspects of erosion using thermal shocks and simple fluid transport simulated as diffusion processes [MKM89, BF02, BF01]. Later approaches focused more on water simulation using either the Eulerian [BTHB06, WCMT07, CMF98] or Lagrangian solution of the Navier–Stokes equations [ASA07, KBKŠ09], and some ap-

proaches even consider wind simulation [KHM\*20]. Other methods considered simultaneous glacial erosion and mountain ridge formation [CJP\*23, AGP\*20], inverse erosion [YCC\*24], and how fauna and flora affect terrains in the long term [ENCC\*21].

**Control:** Simulations and procedural models are challenging to control. Several approaches attempted to address the controllability of terrain models undergoing erosion processes by navigating the fluid simulation [SBBK08], by providing user control over the erosion parameters [JFBB10], or by manipulating continents at large scales [CBC\*16]. Recent methods often abstract physics out entirely [GGG\*13] or combine it with intuitive means to modify terrains, such as brushes [CCB\*18, EVC\*15], diffusion [LGP\*23], gradient domain [GPM\*22], snow [CEG\*18] or style transfer [PPB\*23, RKČ\*22]. Real terrain patches have been used to generate new terrains from sketches [ZSTR07]. This method does not capture large-scale features and requires fine-tuning the patch connections. Our work focused on disentangling the contours to ensure a clean output without undesired contours.

**Terrain Generation using Machine Learning:** Close to our approach is the conditional GANs [HAYC20] for the terrain generation by Guérin et al. [GDG\*17], where the user draws a 2D sketch and the system generates a DEM. The style is encoded through an embedding [ZLB\*19], and a new style can be generated according to the interpolation of existing embeddings. The model is adversarially trained and conditioned on the corresponding embedding, indicating that it only needs one model, and the authors provided different synthesizers for easy terrain authoring. The problem with this method is the need for a large training dataset for a single style. Our algorithm uses a single terrain to generate similar models. Rajasekaran et al. [RKČ\*22] focused primarily on the assessment of the generated terrain and realism terrain, but they also used Cycle-GAN [ZPIE17] to map the data from two domains. More recent work [GPM\*22] considers the gradient instead of elevation in the terrain generation. The benefit of the gradient is that it can



**Figure 2:** Overview: The single input DEM ‘style’ is used to extract its ridges and valleys, which we call a DEM contour. They are organized into a DEM contour graph, and a GNN model is trained. Meanwhile, a SinGAN is trained on high-frequency features of the input respectively. The user draws a sketch, its parts are matched to the DEM style contour, and the trained GNN generates new terrain with high-frequency details added by a SinGAN. The result is a DEM following the user sketch resembling the input. The user sketch is directly input by the users, shown in the first ‘Sketch’ block in the second row. The DEM Contour Parts Extraction block and User Sketch Parts Extraction block are the same algorithm with different inputs.

be seamlessly blended into the original terrain without breaking the formalism. The recent approach of [HHM\*23] claims the previous works based on GAN have a tradeoff between flexible user control and maintaining generative diversity for realistic terrain. They applied diffusion models and proposed a multilevel denoising synthesizer to generate structural and fine-tuned information. Meanwhile, another diffusion-based work [LGP\*23] aims to handle more classes of terrain. The framework can automatically clean the datasets and provide several real-time authoring techniques.

However, the common problem of these methods is the need for a large amount of data. Meanwhile, there is no detail control. For example, if the author only wants one exact ridge, the previous model can not guarantee this. The same sketch in different positions may generate different results. Moreover, the ‘style’ of the terrain is intrinsically determined by the choice of the training dataset. Our approach uses a single image, which defines the style.

**Deep Learning** techniques have shown great success in generative domains, and our work relates directly to two main groups of algorithms: GANs and GNNs. GANs [GPAM\*20] utilize an adversarial training strategy to generate realistic images, and conditional GANs [HAYC20] generate images based on some conditions. However, they need large amounts of data for training. The work proposed in [RSDM19] learns from a single image, but the pipeline is based on CNN, and it is hard to explicitly capture the topology information. Note that [HHM\*23] implicitly preserves the topology by adding the sketch constraint, which needs data while the topology is still not guaranteed. This becomes more severe in the terrain generation. GNN-based algorithms handle the topology information in the data, and the convolution is generalized to the graph domain [KW17]. However, these methods can only model the pairwise relations [XHLJ19]. An intuitive way to generalize the graph concept to the terrain is to consider each pixel a node. Then, modelling higher-order relationships is necessary. A hypergraph [LM17] is an important concept for 3D point clouds [ZLZ\*18], network analysis [LM18], and other domains. It generalizes the pairwise relation, that is, edge, in the original graph setting to the high order relation, that is, hyperedge, which contains multiple nodes. Learnable-

based hypergraph modules are also proposed [FYZ\*18, CPPM22, WYL\*22]. The equivariant property [WYL\*22] is shown to be an important aspect. Since our model focused on detailed topology and geometry, we need a model that can capture the high-order relationship among different nodes. The previous high-order models [FYZ\*18, CPPM22, WYL\*22] focused on the node information only, ignoring the edge information. Our model adds edge information to the training process so that the field can be explored thoroughly. Then we apply a SinGAN [RSDM19] and add a different loss designed explicitly for terrain to add local features.

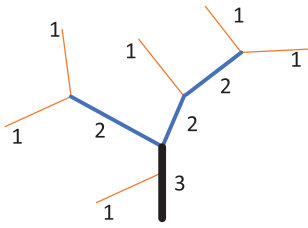
### 3. Overview

Our method consists of three main steps: sketch extraction, training, and inference (see Figure 2).

**Sketch extraction:** We have a DEM contour and a sketch, which are the same data structure under different scenarios. The DEM contour is extracted from the real DEM for training and the other is input by the user during output generation. Sketch is a list of 1D sketch parts that capture terrain valleys and ridges organized into a sketch graph.

Each sketch is divided into small geometric parts 5–10 pixels long, storing its elevation and shape. Later, we match the sketch parts of the input to the user sketches. The length is a compromise between the expressivity and the function. Although longer sketch parts contain more information, it is hard to find a similar long sketch in the original DEM during sketch part matching, which makes the generation unrealistic.

Then, we build the sketch graph assuming the sketches affect their neighbourhood and vice versa. We use the scatter and gather processes to capture the detailed relation by learning a GNN model. This sketch region construction step then builds a graph, capturing the geometric information about the sketches and the remaining terrain parts. We can handle the rotation and transition in the CNN-based models without any further design modules.



**Figure 3:** Strahler ordering.

**Training:** The input is a single DEM defining the overall style of the output. The sketch extraction step captures the main features, but the GNN does not learn small features. We use a CNN-based network SinGAN [RSDM19] to restore the high-level features. SinGAN conducts inpainting, editing, harmonization and super-resolution using the image input. Training SinGAN [RSDM19] takes one single DEM as input and learns how to generate new images.

The **inference** step uses the user sketch graph containing ridges and valleys. We take each sketch part from the user sketch and find the most similar (shape and elevation) part in the DEM contour. We align the sketch parts from the user input with the DEM. The same sketch region construction and GNN model are applied to get a coarse-generated DEM. The SinGAN then automatically refines the details.

#### 4. Training

**Data:** A digital elevation model (DEM) is a discrete grid where each point represents the height. The DEM can be of arbitrary size, but the GPU memory dictates their processing size. Unless explicitly mentioned, we used DEMs of size  $128 \times 128$ . Section 7.5 shows how to generate much higher-resolution results.

##### 4.1. Contour and contour parts extraction

The critical element to user control is the definition of features that are important for user sketches. Similar to the previous work [GDG\*17], our contours encode ridges and valleys. Our model adopts the eight-direction (D8) flow model [JD88] to get the ridges and valleys contours. The D8 flow algorithm defines the water flow of a pixel by choosing the direction with the largest gradient. Each pixel has a unit of water, which is transferred to the lower neighbours according to the gradient. The output of the D8 flow algorithm is noisy due to the randomness in the input terrain, which leads to a noisy contour map. We reduce the noise by four post-processing steps: (1) close operation, (2) finding Strahler stream order [Hor45], (3) skeletonization and (4) filtering.

*The close operation:* The greediness in the D8 algorithm indicates that some parts of the ridges may be disconnected, resulting in intermittent contours. We apply the close operation to connect the contours with several missing pixels.

*The Strahler stream ordering* (Figure 3) was designed to determine the stream order, that is, to distinguish between the main-

stream and channels. The ordering provides the stream hierarchy, and we use it to find the order of the contours as it distinguishes between the essential contour parts and tributaries for ridges and valleys.

*The skeletonization* operation ensures the contour is always a one-pixel wide line while preserving the connectivity and topology of the contours.

*The filtering* is the next step, which deals with noise. Small deviations will generate short random contours, for example, tiny holes on the Moon or the plains beside the Grand Canyon. Our model filters out contours shorter than 10 pixels or contours with an average gradient of each pixel within the lowest 1%. We also split long edges into smaller parts to ease the training.

*Contour Part Extraction* divides the input contour/sketch into smaller contour parts that are then used to match the DEM and the user sketch. The extraction splits the long contours into small contour segments: (1) we start from an endpoint, perform the depth first search (DFS), and store the result in a list. (2) Start from the beginning of the list, we randomly choose 5–10 continuous elements. There are two situations. First, the elements are adjacent in the original DEM; second, the elements are not adjacent in the original DEM. For the first situation, we consider it as a contour part. For user input, it is a sketch part. Then, we remove the elements from the list. For the second situation, we find the elements that are not adjacent to the original DEM and split the long sequence into two short sequences. Short sequences ( $< 5$  pixels) are ignored. The other sequence is considered as a contour part. Then, we remove the detected contour parts from the graph and repeat the process until no elements are left.

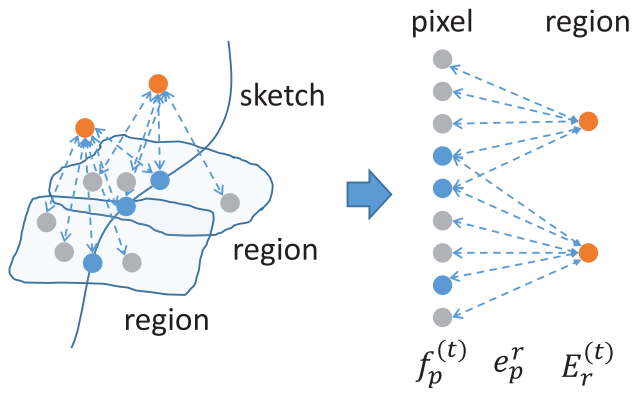
##### 4.2. Contour region construction

One important assumption is that the contours strongly influence the regions around them. The pixels on the contour correlate with those around that contour, and we model this connection using a bipartite graph (Figure 4). The left-hand side shows the nodes corresponding to the pixels. On the right-hand side, the nodes indicate the contour region. For the user input sketch, it is the sketch region.

To construct the graph, we connect pixels to regions. The edges are composed of two parts: fixed and randomly chosen edges. The fixed edges start from all pixels connecting to the  $k$  nearest regions. The randomly chosen edges are randomly sampled from arbitrary pixel-to-region pairs. In our experiments, we set  $k = 10$ , and we randomly choose 10,000 edges for each epoch. Ideally, all the pixels in the region should be aligned to that sketch. However, this would be time-consuming to calculate and train. Instead, we align each pixel to the ten nearest sketches, randomly sample some other sketches, and construct the sketch region.

##### 4.3. The GNN model

The modelling step aims to model a whole area based on the contour (sketch). We assume one contour will only affect the region near it. We first gather the information in that region, then scatter it to



**Figure 4:** The sketch region construction creates a bipartite graph. The left figure shows an original sketch being divided into small sketches. The grey circles indicate all pixels in the region besides the ones on the sketch. The blue circles indicate the pixels on the sketches. The orange circles are the region.  $f_p^{(t)}$ ,  $E_r^{(t)}$  indicates the embedding of the pixels and regions, where  $t$  indicates the iterations. The symbol  $e_p^r$  indicates the relation between a pixel and a region.

the entire region and update the individual pixels. Let us denote the region affected by a contour  $s$  as  $r_s$ .

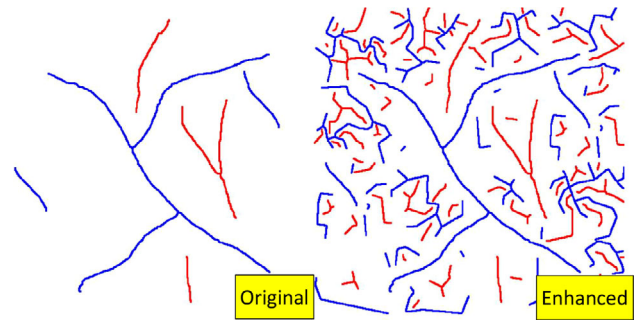
$$\begin{aligned} E_r^{(t)} &= E_r^{(t-1)} + \text{Gathr}_p(\{\text{Enc}_1(f_p^{(t)}, e_p^r) \mid \forall p, p \in r\}, E_r^{(t-1)}), \\ f_p^{(t)} &= \text{Gathr}_r(\{\text{Enc}_2(E_r^{(t)}, e_p^r) \mid \forall r, p \in r\}, f_p^{(t-1)}), \end{aligned} \quad (1)$$

where  $t$  is the iteration index,  $t = 1, 2, \dots, T$ .  $f_p^{(t)}$ ,  $E_r^{(t)}$  represents the pixel and region information for the  $t$ -th iteration,  $\text{Gathr}_p$  and  $\text{Gathr}_r$  are the functions to gather the information. We applied graph attention model [VCC\*18] for  $\text{Gathr}_p$  and  $\text{Gathr}_r$ . Both should be set permutation invariant; thus, we chose the graph attention model. Here, we have  $\text{Gathr}_r$  because one pixel may be affected by different small contours, and  $\text{Enc}_1$  and  $\text{Enc}_2$  are two-layer MLPs with a hidden embedding dimension 32 to encode the information stored in pixels and regions.  $e_p^r$  stores the information for the pixel  $p$  corresponding to the region  $r$ , which is the key part of our model. We discuss how we select and design the  $e_p^r$  in Section A.4.

Our model performs multiple iterations because (1) they construct a deep neural network, which can increase the representation power; (2) multiple iterations construct the correlation among different contours. Equation (1) shows that the information from different contours is gathered from the same region. Then, the region will scatter the gathered information back to the contours again. This promises that the closer sketches, that is, contours that share more regions, will have a stronger information exchange since they share more pixels.

#### 4.4. SinGAN

SinGAN [RSDM19] overcomes the GAN's huge data requirement, enabling it to train and generate samples from a single image. The newly generated images are visually similar to the original image because SinGAN includes a patch similarity discriminator  $D$ , focusing on the details (high-level features). However, it is an uncondi-



**Figure 5:** A user-defined sketch (left) is automatically enhanced with high-frequency information (right).

tional generative model, indicating that it cannot directly be applied to our contour-based task. Thus, by combining our GNN model and SinGAN, we exploit the advantage of the two algorithms and generate realistic terrain containing high-level and low-level features. We adopt SinGAN in two ways. First, the SinGAN takes the three RGB channels, while DEM only has one. Second, since the gradient is necessary for the terrain, a loss function  $l_{\text{gradient}}$  is added to SinGAN in the reconstruction loss part (see Section 6.1). The detailed steps are as follows: after the GNN step, each pixel will have an initial height. We resize the initial height into a 2D array (images) and then feed this image directly to the modified SinGAN. SinGAN is *not* a conditional generation method. By applying SinGAN, the model will never learn how to map contours to the height map and generation from contours.

## 5. Inference

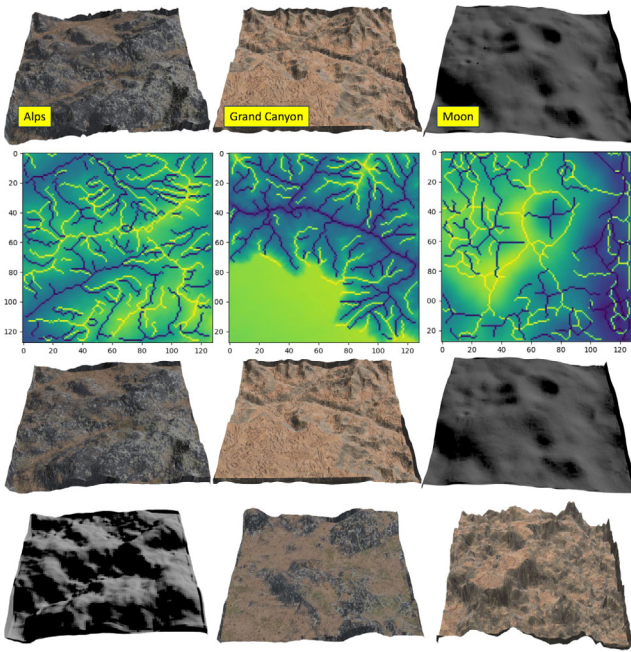
### 5.1. User input sketch

Our model enables both user-defined sketches and sketches based on features extracted from DEMs using the method described in Section 4.1. The user draws ridges and valleys with selected heights. The height starts from a user-defined value and its height either gradually decreases or increases. After that, the pixels with height are then normalized, and our model generates visually similar terrain based on the sketches.

### 5.2. Random sketches

One essential assumption in machine learning is that testing data should come from the same distribution as the training data. However, user-defined sketches do not necessarily follow the same distribution. Usually, the user sketches are several simple curves, while the training sketches are dense, as seen in Figure 5 left for user input and Figure 6 second line for DEM contours. Inspired by the idea of the dividing tree generation algorithm from [AGP\*19], we enhance the user sketch with small random sketches.

The random sketch (see Figure 5) generation process includes the following steps. We randomly sample  $n$  nodes and assign them a value sampled from  $[0.5, 1]$  as the height. We then find the Minimum Spanning Tree (MST) [Gab77] among these  $n$  nodes. The MST is considered the potential ridge. We then triangulate the  $n$



**Figure 6:** The 3D rendered samples of the real-world DEMs (top). Their automatically extracted features (the second line) are used as a sketch to ‘reconstruct’ them (the third line). We exchange the sketches and show the ‘reconstruction’ (bottom). The Moon style is applied to the Alps sketches, the Alps style to the Grand Canyon sketches and the Grand Canyon style to the Moon sketches. The last row is the result generated using the same sketch but with different styles. This shows that the output is controlled not only by the sketch but also by the input image.

nodes with the Voronoi Diagram and randomly assign a value sampled from  $[0, 0.5]$  as the height of each node. We then find another MST, this time in the vertices of the Voronoi Diagram [AK00], and consider it the potential valley. We then select edges in potential ridges that do not cross the user-defined ridges. Similarly, we select the edges in potential valleys with no cross with the user sketch and the chosen ridges as the valleys. We then connect the ridges and valleys and use linear interpolation to find the height among the edges, similar to [KMN88] (see Figure 5).

### 5.3. Sketch part alignment

The user sketch is divided into sketch parts using the same algorithm for contour part extraction for DEMs (Section 4.1). We have a specific embedding for each contour part in the training phase but no embedding for the inference phase. Thus, we need to assign an embedding to each sketch part. The sketch part alignment uses shape and height information. Overall, we want to find the contour part most similar to the ones in the user input. To compute the shape (including the rotation), we first calculate the gradient of the position  $(x, y)$ . Then, we rotate it so that the main direction is the same. Then, we apply the longest common substring (LCS) algorithm [Gus97] to choose the top 10 candidates. Finally, we check the height infor-

mation (mean and variance) to select the final round of candidates and randomly choose one.

The overall algorithm is as follows. Each sketch part is a set of consecutive points  $[P_x^i, P_y^i, P_z^i]$ . Our algorithm first rotates the sketches and contours to align them in the main direction (see details in Algorithm A1). The rotation angle can only be  $k\pi/4$ , where  $k \in \mathbb{N}$ . After rotation, the vectors that start from the first point and end at the last point should have an angle  $\leq \pi/4$ .

We then choose the contours with a similar shape projected to 2D, thus only considering the  $P_{i_x}$  and  $P_{i_y}$  coordinates using the LCS algorithm (see details in Algorithm A2). We then align the  $P_{i_z}$  coordinates by filtering the contours with large height differences on the common substrings. In particular, we calculate the LCS of two normalized point lists. Assume the original curve is  $\{(P_x^1, P_y^1), (P_x^2, P_y^2), (P_x^3, P_y^3), \dots, (P_x^n, P_y^n)\}$ . The normalization is

$$\left\{ \frac{(P_x^2 - P_x^1, P_y^2 - P_y^1)}{\|(P_x^2 - P_x^1, P_y^2 - P_y^1)\|_2}, \dots, \frac{(P_x^n - P_x^{n-1}, P_y^n - P_y^{n-1})}{\|(P_x^n - P_x^{n-1}, P_y^n - P_y^{n-1})\|_2} \right\}.$$

Since all the contours are connected, the normalization should contain eight elements only  $\{(\cos(k\pi/4), \sin(k\pi/4))\}$ , where  $k = 0, 1, \dots, 7$  defines the Euclidian distance between two elements  $u$  and  $v$ .

We then find the sketch with the smallest height difference on the longest common substring detected in the previous part.

### 5.4. Generation

After we have the user-designed sketches and corresponding indices, we apply the contour (sketch) region construction process used in the training part (Section 4.2) to construct the graph. Then, our GNN model and SinGAN are applied to generate the final result.

Although our model chooses the most similar contour index, they are not the same, and the generated DEM can be discontinuous due to the sampling issue and lack of data. Inspired by DeepSDF [PFS\*19], we address this by a fine-tuning step before the generation, ensuring that (1) the contours are similar to the user-designed ones and (2) the generated DEM is smooth. We freeze all the parameters in the model except the embedding of contours index  $E_{md}(\cdot)$ . Then, our model updates the embedding of the contour index to minimize the  $l_{gt} + l_{smooth}$ . Here,  $l_{gt}$  only calculates the pixels on the sketches, and  $l_{smooth}$  calculates the general smoothness in the generated terrain. Finally, SinGAN is applied to provide high-frequency features.

## 6. Implementation

We implemented our system in Python with Pytorch 1.11 and PyG 2.1.0 for the training. We used an Nvidia V100 GPU with 16GB memory and ran it on a cluster with an AMD EPYC 7543 32-core CPU. The GNN training on  $128 \times 128$  image as input takes about 3.5 min for 1000 epochs and the learning rate is 0.0001. We trained over 60,000 epochs to ensure our model could fit the given DEM. The SinGAN was run with the same settings as [RSDM19], and it takes around 30 min to train one image. The inference process takes

around 20 s to generate one  $128 \times 128$  image. The UI is written in Python Tkinter. We used Blender on a laptop with an i9-13900HX CPU, 16GB memory, and a GeForce RTX 4060 GPU with 8GB of memory to render the final images. The training is not memorizing the image, but it disentangles them, which is a harder task that takes longer. Secondly, we apply GNN to provide a more flexible representation of our model. GNN takes much more computation resources as a tradeoff and is slower than CNN.

Our implementation cannot generate terrains of size  $512 \times 512$  or larger due to the limitation of the GPU memory size. To generate large terrains, we split large terrains into  $128 \times 128$  patches and generate. We combine the results by interpolating the edges. For example, to generate a  $512 \times 512$  image, we first split the sketch into 16 images and generate 16  $128 \times 128$  images. This takes around 5 min, with every patch needing around 20 s. The bottleneck of the inference is the time to load the model and data for each image. Please note there are additional technical details about the implementation in the Appendix.

### 6.1. Loss functions

Our model is end-to-end aiming to solve the following function

$$\min_{\text{Emd, Enc, MLP}} l_{\text{total}}, \quad (2)$$

where  $l_{\text{total}}$  is a single six-component loss function:

$$l_{\text{total}} = l_{\text{gt}} + l_{\text{ssim}} + l_{\text{gradient}} + l_{\text{sketch}} + l_{\text{smooth}} + l_{\text{further}} \quad (3)$$

We applied different weights for different losses, but the results remained similar, although they have different scales. This is because  $l_{\text{gt}}$ ,  $l_{\text{gradient}}$  and  $l_{\text{further}}$ , controlling the exact values of each pixel, are all very small. Meanwhile,  $l_{\text{smooth}}$  and  $l_{\text{sketch}}$  aim at the disentanglement between ridges and valleys.

The first component of the loss function aims to make the model fit the terrain given the DEM information:

$$l_{\text{gt}} = \sum_p \|\tilde{y}_p - y_p\|^2. \quad (4)$$

The second term in reconstruction tasks is SSIM [WBSS04] loss  $l_{\text{ssim}}$ . Here, we also applied the SSIM loss to our training pipeline to ensure the reconstructed terrain was structurally similar to the original one. Since the gradient is also important in the terrain generation, we also apply the third gradient loss

$$l_{\text{gradient}} = \sum_{\forall p} \sum_{p' \in \mathcal{N}(p)} (\|\tilde{y}_p - \tilde{y}_{p'}\|^2 - \|y_p - y_{p'}\|^2)^2.$$

Ridges are convex, while valleys are concave, but there is no guarantee that a neural network will naturally learn this, especially when it has limited data. Therefore, we propose a heuristic loss function  $l_{\text{sketch}}$  as follows: give an arbitrary point, we first project to its neighbouring contours. We need the height of that point to be larger than the projections to the nearby valleys and smaller than the projections to the nearby ridges. Since the comparison is not differentiable, we apply the Gumbel-Softmax Trick to have a surrogate function [JGP17]. A cross-entropy loss  $CSE$  is applied after sampling

from the larger height using the Gumbel-Softmax. Here we denote the as  $\mathbb{I}_p, \mathbb{I}_p = [1, 0]$  if  $p$  is ridges;  $\mathbb{I}_p = [0, 1]$  if  $p$  is valley.

$$l_{\text{sketch}} = \sum_p CSE(\text{Gumbel-Softmax}(\tilde{y}_p, \tilde{y}_{\text{proj}_p}, \mathbb{I}_p)) \quad (5)$$

The fifth component of the loss function is similar to  $l_{\text{sketch}}$ . Instead of maintaining the convex or concave property, this loss focuses on the smoothing. Our model will sample pixels  $p'$  and  $p''$ , where  $D(p', p'') < 5$  and minimize the following objective function, where  $D(\cdot)$  indicates the Hamming distance:

$$l_{\text{smooth}} = \sum_{p', p''} |\tilde{y}_{p'} - \tilde{y}_{p''}|.$$

The last component of the loss function ensures that further distances lead to a smaller affection. We sample the pixels  $p'$  with long distances to a contour. The loss function is to minimize:

$$l_{\text{further}} = \left\| \left\| \tilde{y}_{p'} - \text{MLP}_3(f_p^{(0)}) \right\| - \frac{1}{10 \times \text{dist}_{p'}^2} \right\|^2$$

Note that when calculating the  $l_{\text{sketch}}$ ,  $l_{\text{smooth}}$  and  $l_{\text{further}}$ , we need to calculate it separately for ridges and valleys. However, heuristically, we found that our model still performs well when we apply these loss functions directly to the final layer  $f_p^T$ .

## 7. Results

To show that our model can learn a single image and generate similar terrain, we show how the preprocessing part finds the sketches, the model's expressive power, generalizability and additional examples.

### 7.1. Datasets

We experimented with seven different datasets: the Alps, the Grand Canyon, the Amazon River, the Sahara, the Danakil Depression, the Himalayas, and the lunar surface (see the first row in Figure 6 and Figure A1). The Alps are geologically fresh areas with large valleys and ridges. The Grand Canyon is widely eroded, with deep valleys and flat areas. The Amazon River area contains river channels but is relatively flat. The Danakil Depression is elongated in shape, flat in the plain, and uneven in the remaining area. The Sahara is smooth and has landforms eroded by wind. The Himalayas are geologically fresh and look similar to the Alps, with smooth and large ridges and valleys. The lunar surface is monotonous, with local peaks and dips.

The six Earth datasets come from the Geospatial-Intelligence Agency (NGA). The maps were provided by the Shuttle Radar Topography Mission (SRTM) [FRC\*07], which scanned the areas in February 2000. The original data with one arc-second resolution (around 30 metres per pixel) have been edited by the NGA to fill small voids and remove spikes and wells. Larger voids were filled using interpolation algorithms.

The Moon DEM comes from the United States Geological Survey (USGS). The Lunar Orbiter Laser Altimeter (LOLA) [YNGS\*08]

provides a precise global lunar geodetic grid. The resolution is 10 metres per pixel.

The DEMs provide only the terrain height. We textured and rendered the DEMs to convey the visual feel and look of the corresponding areas. While some figures in this paper are textured, we also show some as a bare elevation with diffuse lighting.

## 7.2. Sketches in patches 128 × 128

**Extracted Features and ‘Reconstruction’:** The second line of Figure 6 shows the ridges (yellow lines) and valleys (blue lines) extraction for several terrains. Using the extracted edges, we can use our algorithm to ‘reconstruct’ the input terrain as shown on the third line of Figure 6. The similarity between the input and the generated terrain is captured by the similarity loss  $l_{ssim}$  from Equation (3). Note that the value of  $l_{ssim} \rightarrow 0$  indicates dissimilar data, and  $l_{ssim} \rightarrow 1$  shows high similarity. The  $l_{ssim}$  for the Alps, Grand Canyon and Moon are 0.97, 0.97 and 0.99 respectively. We did not expect a pixel-exact copy of the input. Our approach generates slightly smoother terrains because the sketches do not consider all features, for example, the small craters on the Moon. Also, the SinGAN injects noise into the generation process.

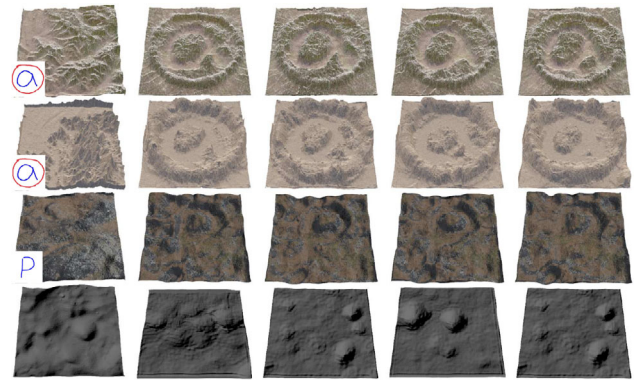
We exchanged the sketches for each terrain and generated the corresponding terrain at the bottom of Figure 6. We apply the Moon style to the Alps sketches, the Alps style to the Grand Canyon sketches, and the Grand Canyon style to the Moon sketches. The results show that our model captures the main features and transfers them to the new terrain while maintaining the same trend in the original terrain.

The **User Sketches** are shown in the first three lines in Figure 7 to demonstrate our model’s diversity and fidelity.

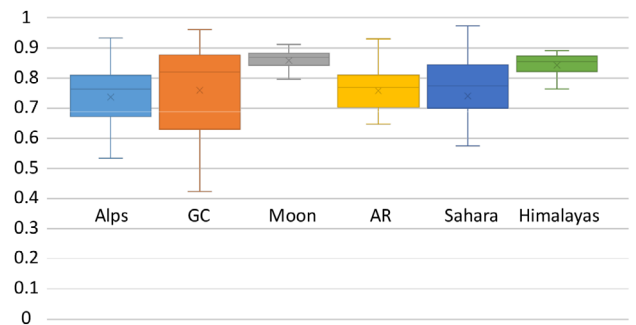
An important property of the generation model is its *diversity* which comes from four parts: (1) Sketch Region Construction in Section 4.2, (2) Random Sketches in Section 5.2, (3) Sketch Alignment in Section 5.3 and (4) Perturbation on  $e_p^r$  and  $Emb(Idx_{sketch})$  in Section A.4. Our images show that all these different parts can contribute to our image diversities while following our user-defined sketches.

We also show generated Moon-style terrain based on the real-world sketches on the bottom line of Figure 7. Most of the generated terrains follow the general trend of the ground truth. However, one main problem is the smoothness. Since our model is at a low resolution, the SinGAN for enhancing high-level features may add too many sharp features. Also, some depressions are not captured by our model. This is mainly due to sketch extraction. Since we only capture valleys and ridges, no specific depressions are captured, but they may be added by drawing an ‘x’.

**Fidelity:** Figure 8 shows how similar terrain our model can generate given some real-world sketches. We apply the SSIM [WBSS04] to calculate the similarity. Although other methods, such as [RKC\*22], define how well a terrain is generated, these are metrics designed for the generation without ground truth. Because we do have ground truth, the SSIM provides sufficient results. We follow the same experiment setting: pick one represen-



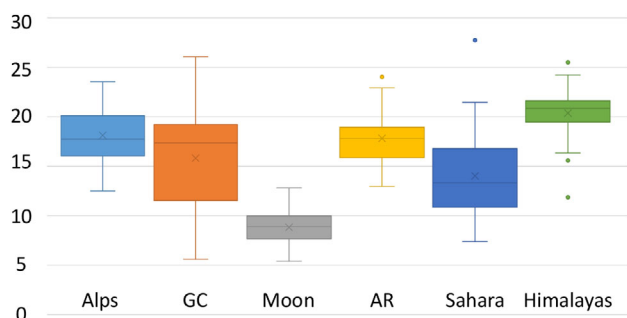
**Figure 7:** Different generated terrain. The first column is the training input, including (from top to bottom), that is, the Amazon River, Danakil Depression, Alps, and the Moon. For the Amazon River and Danakil Depression, we use the sketch directly, without any amplification. The outer part is a circle ridge while the inside is a ‘a’ valley. The diversity comes from the Sketch Region Construction and Sketch Alignment. We have a letter ‘P’ indicating Valley for the Alps, and our model automatically enhanced the sketch with additional features. The diversity comes from the Random Sketches and Perturbation on  $e_p^r$  and  $Emb(Idx_{sketch})$ . We follow the sketches from real-world scenarios for the lunar surface. All images have the same 128 × 128 resolutions. The first three rows’ user sketches are shown in the bottom left corner of the first column. The last row does not share the same sketch since they are from different patches.



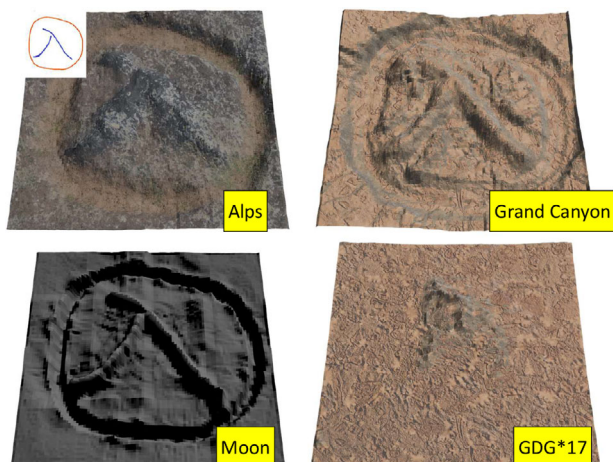
**Figure 8:** SSIM loss of real-world terrain. SSIM is a common metric for image restoration. SSIM ranges from  $-1$  (anticorrelated) to  $1$  (perfectly similar), and  $0$  indicates no similarity. GC stands for the Grand Canyon, while AR stands for the Amazon River.

tative 128 × 128 image as input and generalize to the remaining area. We skipped the Danakil Depression since most of the area is blurred and contains little information. Our model performs best on the Moon and Himalayas since the sketches are self-similar. The variance of the Grand Canyon is large because some patches do not share the same pattern, containing very few shallow valleys. The Alps performs the worst among all these datasets but still has an average SSIM of 0.75. One major problem is that the target terrain is only distributed on a corner or half of the image in many patches. The heavily unevenly distributed sketches will negatively affect the generation since we have no information for the blank area.





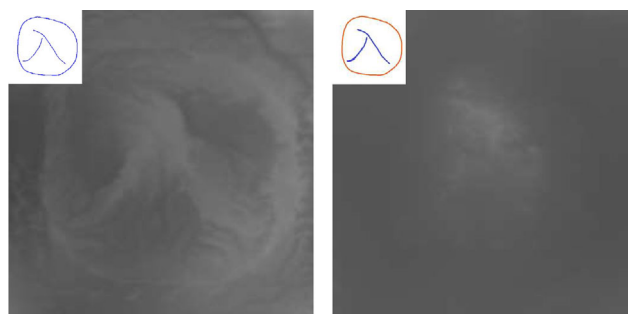
**Figure 9:** PSNR loss of real-world terrain. PSNR is a metric for image reconstruction. The unit of PSNR value is dB. The larger value indicates a more accurate reconstruction. GC indicates the Grand Canyon, and AR stands for the Amazon River.



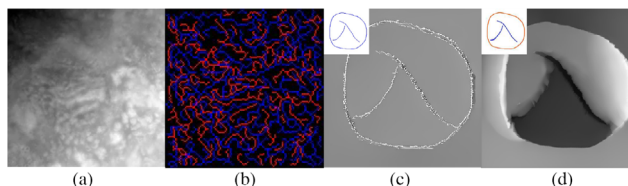
**Figure 10:** Comparison of our model and [GDG\*17].

Meanwhile, the missing ridge and valley sketches in the sketch extraction process also make it difficult to reconstruct the terrain fully. We also report our PSNR value for our real sketch reconstruction task in Figure 9. PSNR is a commonly used measurement of the reconstruction quality of models based on sketches. The higher value indicates a better reconstruction quality. In our experiments, the Moon has a lower average reconstruction quality, probably because the Moon has denser ridges and valleys, and cutting it into patches hinders the full representation of the Moon's reconstruction. The real sketches indicate that our model can regenerate realistic terrain.

Because most of the previous work is based on the [GDG\*17], we **compare** our method in Figure 10. The figure has a ridge sketch with the  $\lambda$  shape and a circle valley sketch outside. We reproduce the result of [GDG\*17] by following the personal website and applying the pre-trained model to the Pix2pix [ZZE17] model in Figure 10. The generation of [GDG\*17] does not satisfy our user-defined sketch and contains too many redundant sketches. This is likely because a predefined elevation map is needed for better generation. More discussion is in the next experiment.



**Figure 11:** Detailed analysis of why [GDG\*17] fails.



**Figure 12:** Comparison of ControlNet [ZRA23] finetuned by USA dataset. (a) and (b) are the training DEM and the corresponding training input sketches. (c) and (d) are the results using the same input sketch from Figure 11.

We continue the following experiment in Figure 11 to compare with [GDG\*17]. Note that two  $\lambda$ s are different. The left is similar to the original paper [GDG\*17] where all sketches are ridges. However, our experiment has a valley around it. The result shows that when we have all sketches as ridges (the same as in the original paper), we get a reasonable output DEM. However, the model fails to generate a plausible result when we put ridge and valley sketches together in a less common way (the valleys are independent and have no interaction with the ridge). This shows that the CNN-based method does not disentangle different features and does not generalize to out-of-distribution situations.

### 7.3. Comparison with ControlNet

Recently, diffusion-based models have shown outstanding performance in image generation tasks. We compare our method with the diffusion-based model ControlNet [ZRA23]. We finetuned the ControlNet model based on the 7000 DEMs sampled from the United States and applied the sketch parts extraction algorithm Section 4.1 to extract the DEM contours. The input of the ControlNet model is the DEM contours of ridges and valleys, and a prompt, 'Given the ridges and valleys, please create the Digital Elevation Model'. The output initially consists of RGB images, but we transfer them into grayscale images. We use a V100 to finetune the ControlNet based on the backbone of Stable Diffusion 2.1 [RBL\*22], which takes 5 days for 500 epochs. Then, the same  $\lambda$  shape with ridges and valleys is fed into the model. The result is shown in Figure 12. Similar to the work [GDG\*17], the method suffers from the need for a large amount of training data, slow training speed, and still cannot disentangle the ridges from the valleys and generate a bad result when the conditional input is clearly out of distribution. Our proposed models

**Table 1:** PSNR values (dB) baselines for a real-world scenario (higher value is better).

Dataset	Alps	Grand Canyon	Amazon River	Sahara	Himalayas
Stable diffusion	8.33 ± 0.88	8.33 ± 1.48	8.48 ± 0.84	7.69 ± 1.72	7.92 ± 1.02
Ours	18.12 ± 2.78	15.81 ± 4.99	17.82 ± 2.58	14.02 ± 4.47	20.38 ± 2.42

are more efficient (4 h compared to 5 days) and have better generalizability. For inferring, ControlNet takes around 20 s for each patch, which is similar to our model.

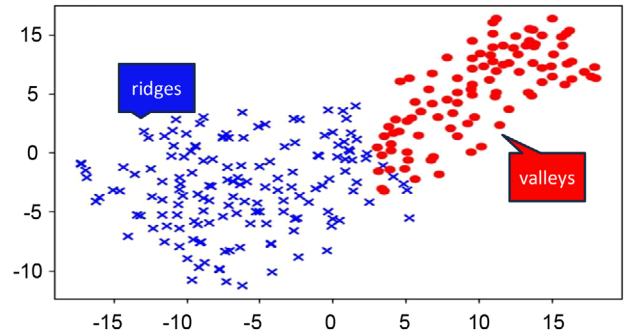
We also conducted experiments to show some quantitative results comparing our model to the Diffusion-based models with results shown in Table 1 (around 10dB for PSNR [HZ10] on average). The baseline is a ControlNet [ZRA23] using Stable Diffusion 2.1 [RBL\*22] finetuned by 7000 terrains sampled from the United States. We get the DEM from Commission for Environmental Cooperation (CEC). Note that ControlNet takes much more data than our model (7000:1) and also significantly more time for training (1 week vs. 4 h) on the same hardware. The evaluation metric is PSNR, which is a common metric for image reconstruction tasks. We did not use the FID [HRU\*17] score because the amount of testing sets is limited. The FID requires thousands of images due to their underlying assumption that features follow a multivariate Gaussian distribution. Since our task focused on single-shot terrain sketching, our testing sets are the neighbours (around 49) of the training image.

#### 7.4. Deep neural model

**Generalizability:** For machine learning models, generating from user-defined simple sketches suffers from a significant technical challenge: the input sketches differ notably from those in training. Unlike the densely detailed sketches typically found in training data, these user-generated sketches are sparse, featuring smoother height representations due to the constraints of the user interface design. Furthermore, the user sketches deviate from natural terrain patterns. All these factors can hinder machine learning-based terrain generation models from producing realistic terrains. Even though previous machine learning-based models yield visually appealing results, they lack explanation. Meanwhile, the traditional models do not have the same problem. They ensure the generated output shares the same distribution by leveraging physically informed differential equations.

Our model borrows the idea from the traditional method and endeavours to simulate the traditional terrain diffusion model using neural networks. The key idea is to disentangle the ridges and valleys so that the authors can arbitrarily combine them. We conducted two experiments, *t-SNE* and *partial sketches*, to show that our model can disentangle the sketches. We only show the result for Grand Canyon data due to the limited space, but others work similarly.

*t-SNE* [vdMH08] visualization in Figure 13 shows that the sketches embedding after our GNN model, that is,  $E_r^{(t)}$ , are clearly separated into two groups. Few sketches are hard to disentangle, which is reasonable since the terrain is noisy.

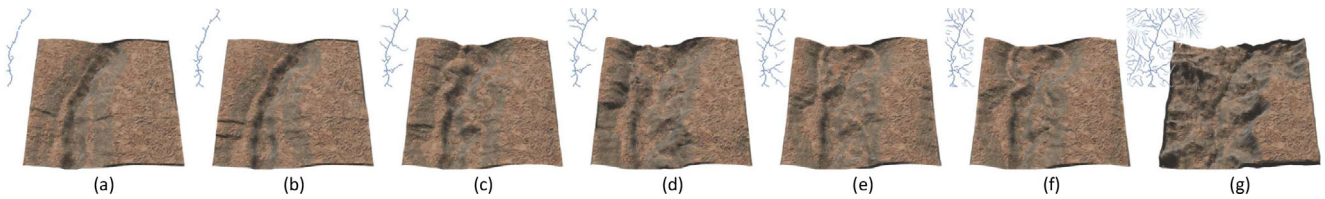
**Figure 13:** *t-SNE* of Sketches Embedding trained on the Grand Canyon.

**Partial Sketches:** In the second experiment, we gradually added more detailed sketches to the terrain. This experiment shows how our model disentangles different sketches and then merges them. Our Grand Canyon data shows one dominant valley across the entire terrain. Besides, it is also affected by the nearby ridges and valleys. The first image in Figure 14 only has the main valley sketch. More and more details are added to the terrain when more sketches are added in the following images. This experiment also indicates when our Sketch Alignment (Section 5.3) process does not work as expected, we can manually change it to some specific alignment when we want the terrain to have the exact same features as shown in the style terrain.

**Topology:** As we claimed in the previous part, our model can preserve the low-level topology information (rotation invariant) since we only consider the relative position between a sketch and its corresponding neighbours. Therefore, when we rotate our sketch for different angles and transit it to different positions, the result in Figure 15 remains very similar. This also proves that our model is robust. When we rotate the sketches, they will be affected by little tweaks due to the 2D image rotation algorithms.

#### 7.5. High resolution

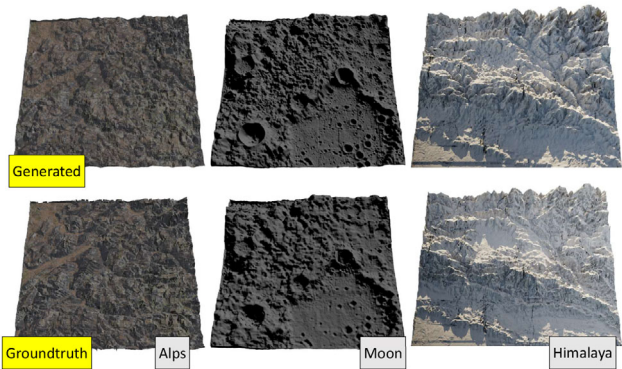
All previous examples were shown on patches of  $128 \times 128$ . Here, we show how we can generate terrains with larger resolutions, for example,  $1024 \times 1024$ . Our model can be applied to different resolutions. We generate all patches separately and then merge them while blurring the boundary. The merging algorithm is simple: first, the DEM is scaled by a ground truth factor since it is normalized to  $[0,1]$  scale for training stability. Then, the linear interpolation for the four rows/columns is applied for each of the two adjacent batches. We show three datasets, that is, Alps, Moon, and Himalayas, in



**Figure 14:** The result of specific sketches. The blue lines on the top-left corner indicate the sketches we applied to reproduce the terrain. From (a) to (e) are the results for valley sketches only, (f) shows the results for both ridges and valleys and (g) is the ground truth. For (a) to (f), we only apply the sketches lie in the left part ( $30 \leq x \leq 50$ ).



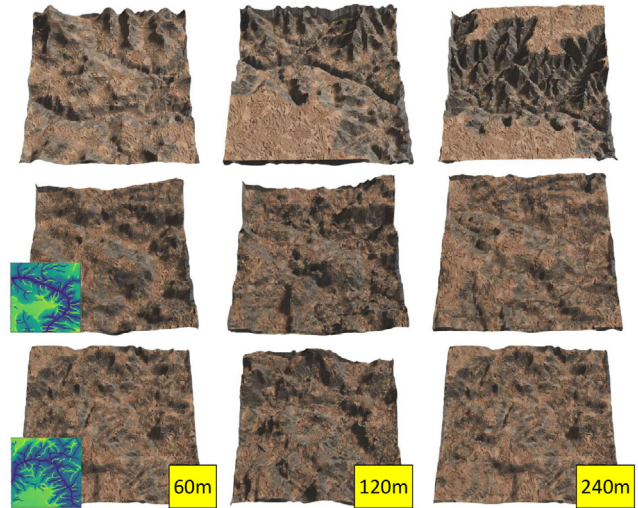
**Figure 15:** The results for different rotation angles and positions. The top is the rendered effect of the Sahara Desert. The bottom is the rendered effect of the Grand Canyon. Note that there will be slight differences in the edge of the image. Our resolution is limited, and the edge will affect the final result.



**Figure 16:** The high-resolution results for different styles.

Figure 16. Some results have a checkerboard effect. Theoretically, our work should not have a checkerboard effect, at least for the GNN part. Since the GNN directly models the relationship of each pixel, the model itself is not limited by the resolution. There are two reasons for that. (1) The GPU memory size limits the resolution. Larger resolution indicates more pixels and complex relationships, making the GNN too large for a 24 GB GPU and time-consuming. (2) We apply a SinGAN for better details, making it hard to generalize directly to a large-resolution image.

We also conduct experiments for different resolutions on the Grand Canyon dataset to show how our model will learn under different resolutions. The result is shown in Figure 17. The lower reso-



**Figure 17:** The results for the Grand Canyon under different scales. The above is the ground truth for our model to learn. The under DEMs are our generation results. All DEMs are  $128 \times 128$  pixels. However, each pixel is 60m, 120m and 240m in three columns respectively.

lution does not include enough details, leading to a smoother result than the DEMs in the 60m resolution DEM, which provides a more noisy detailed output.

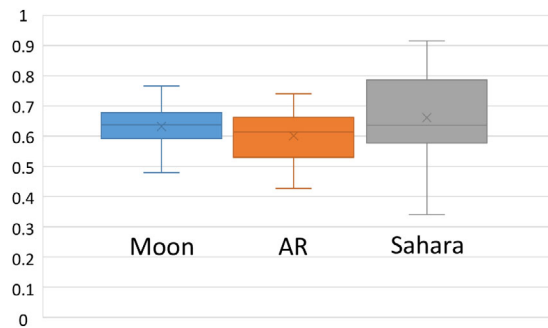
### 7.6. Ablation study

Our paper proposed two critical components: Sketch Part Extraction (Section 4.1) and Random Sketches (Section 5.2). We conducted ablation studies on these two modules.

During the training, our model would suffer from reconstructing the DEM if the sketch part consists of too many nodes (more than 20). A larger sketch part will aggregate more information about neighbouring pixels. The pixel information is then smoothed, and the generated result loses details. Even larger sketch parts, for example, the entire sketch as a sketch part will cause our model to reconstruct the DEM without any details. Figure 18 shows the result when the sketch part consists of 5 – 10 and 15 – 20 pixels for the Sahara Desert and the Amazon River. Examples with 5 – 10 pixels include more details while increasing the size to 15 – 20 loses information.



**Figure 18:** The ablation study results in each sketch part's number of pixels.



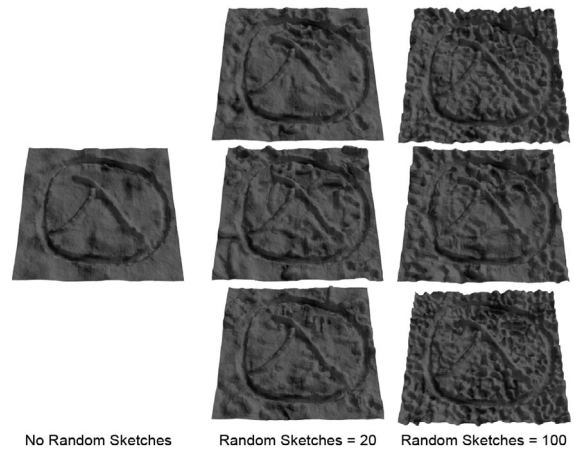
**Figure 19:** The SSIM value after we have the number of pixels of each sketch part around 15 – 20. The average results are lower than the previous 5 – 10 SSIM value, indicating less similarity.

Thus, we choose the 5 – 10 pixels in our setting. This finding also applies to real-world sketch scenarios since in Figure 19, the average SSIM value is lower than in Figure 8. This phenomenon should be alleviated by increasing the representation power of our model. However, we cannot conduct the experiments due to the limitation of GPU memory size.

To prove our Random Sketches component is useful, we show the generated DEM without sketches in Figure 20 first column. Considering that the Moon is composed of craters, the generated smooth terrain does not capture the craters' features. To this end, we apply the random sketches component in the second and third columns of Figure 20. Apparently, if there are too many randomly generated sketches, the generated terrain will suffer from the noise, which leads to a more plausible terrain. A suitable amount of random sketches makes the terrain more realistic. Meanwhile, the randomness of sketches ensures diversity.

## 8. Conclusions and Future Work

Terrain generation using deep learning has been more and more prevailing. In this work, we proposed a new model that merits the ben-



**Figure 20:** The results of the ablation study of the random sketches. The first column is the result of the Moon with a 'λ' sketch only. The second and third columns are the results with different random sketch numbers.

efits of deep learning's expression power while maintaining the loss of topology information. Meanwhile, due to the disentanglement of different features, our model has a better generalization power and can be applied to arbitrary user sketches. Last but not least, our model only needs a single terrain image for the training process and can give a more subtle definition of what style is. Despite these advantages, our model is memory-consuming because all the edge information is stored in the GPU during training. This limits our model from generating larger resolution terrain. More sketch types can be added too, for example, holes on the Moon, rivers, and cliffs. More detailed sketch information will create a more detailed image. Our model can also be generalized to the 3D version for materials or meshes in the future.

*Limitations and Future Work:* One limitation of our approach is that we only use ridges and valleys. Different features could be used to describe the terrain in a more detailed way, similar to [GDG\*17]. Another limitation is the long training time. It takes 4 h to train, mainly because the feature graph is dense. We could decrease the size of the graph by splitting the long sketch into fewer sketches while increasing the neural network size to ensure the representation power. Alternatively, we could use a smaller model, but the precision of the proposed algorithm would suffer. Methods using model quantization and simplification could also be used. Moreover, new generations of GPUs will likely reduce the processing time.

## Acknowledgements

The authors have nothing to report.

## References

[AAC\*17] ARGUDO O., ANDUJAR C., CHICA A., GUÉRIN E., DIGNE J., PEYTAIE A., GALIN E.: Coherent multi-layer landscape synthesis. *The Visual Computer* 33, 6 (2017), 1005–1015.

- [AGP\*19] ARGUDO O., GALIN E., PEYTAIE A., PARIS A., GAIN J., GUÉRIN E.: Orometry-based terrain analysis and synthesis. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–12.
- [AGP\*20] ARGUDO O., GALIN E., PEYTAIE A., PARIS A., GUÉRIN E.: Simulation, modeling and authoring of glaciers. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–14.
- [AK00] AURENHAMMER F., KLEIN R.: Voronoi diagrams. *Handbook of Computational Geometry* 5, 10 (2000), 201–290.
- [ASA07] ANH N. H., SOURIN A., ASWANI P.: Physically based hydraulic erosion simulation on graphics processing unit. In *GRAPHITE '07: Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia* (New York, NY, USA, 2007), ACM, pp. 257–264. <http://doi.acm.org/10.1145/1321261.1321308>.
- [BF01] BENES B., FORSBACH R.: Layered data representation for visual simulation of terrain erosion. In *SCCG '01: Proceedings of the 17th Spring Conference on Computer Graphics (USA, 2001)*, vol. 25(4), IEEE Computer Society, pp. 80–86.
- [BF02] BENES B., FORSBACH R.: Visual simulation of hydraulic erosion. *Journal of WSCG* 10, 1 (2002), 79–86.
- [BTHB06] BENES B., TĚŠÍNSKÝ V., HORNYŠ J., BHATIA S. K.: Hydraulic erosion. *Computer Animation and Virtual Worlds* 17, 2 (2006), 99–108. <https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.77>.
- [CBC\*16] CORDONNIER G., BRAUN J., CANI M.-P., BENES B., GALIN E., PEYTAIE A., GUÉRIN E.: Large scale terrain generation from tectonic uplift and fluvial erosion. *ACM Transactions on Graphics* 35, 2 165–175. <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12820>.
- [CCB\*18] CORDONNIER G., CANI M.-P., BENES B., BRAUN J., GALIN E.: Sculpting mountains: Interactive terrain modeling based on subsurface geology. *IEEE Transactions on Visualization and Computer Graphics* 24, 5 (May 2018), 1756–1769. <https://doi.org/10.1109/TVCG.2017.2689022>.
- [CEG\*18] CORDONNIER G., ECORMIER P., GALIN E., GAIN J., BENES B., CANI M.-P.: Interactive generation of time-evolving, snow-covered landscapes with avalanches. *Computer Graphics Forum* 37, 2 (2018), 497–509. <https://doi.org/10.1111/cgf.13379>.
- [CJP\*23] CORDONNIER G., JOUVET G., PEYTAIE A., BRAUN J., CANI M.-P., BENES B., GALIN E., GUÉRIN E., GAIN J.: Forming terrains by glacial erosion. *ACM Transaction on Graphics* 42, 4 (July 2023), 1–14. <https://doi.org/10.1145/3592422>.
- [CMF98] CHIBA N., MURAOKA K., FUJITA K.: An erosion model based on velocity fields for the visual simulation of mountain scenery. *The Journal of Visualization and Computer Animation* 9 (1998), 185–194.
- [CPPM22] CHIEN E., PAN C., PENG J., MILENKOVIC O.: You are allset: A multiset function framework for hypergraph neural networks. In *International Conference on Learning Representations* (2022). [https://openreview.net/forum?id=hpBTiv2uy\\_E](https://openreview.net/forum?id=hpBTiv2uy_E).
- [ENCC\*21] ECORMIER-NOCCA P., CORDONNIER G., CARREZ P., MOIGNE A.-m., MEMARI P., BENES B., CANI M.-P.: Authoring consistent landscapes with flora and fauna. *ACM Transactions on Graphics* 40, 4 (2021), 1–13. <https://doi.org/10.1145/3450626.3459952>.
- [EVC\*15] EMILIEN A., VIMONT U., CANI M.-P., POULIN P., BENES B.: Worldbrush: Interactive example-based synthesis of procedural virtual worlds. *ACM Transactions on Graphics* 34, 4 (July 2015), 1–11. <https://doi.org/10.1145/2766975>.
- [FFC98] FOURNIER A., FUSSELL D., CARPENTER L.: Computer rendering of stochastic models. *ACM Transactions on Graphics* 17, 4 (1998), 189–202. <https://doi.acm.org/10.1145/280811.280993>.
- [FRC\*07] FARR T. G., ROSEN P. A., CARO E., CRIPPEN R., DUREN R., HENSLEY S., KOBRICK M., PALLER M., RODRIGUEZ E., ROTH L., SEAL D., SHAFFER S., SHIMADA J., UMLAND J., WERNER M., OSKIN M., BURBANK D., ALSDORF D.: The shuttle radar topography mission. *Reviews of Geophysics* 45 (May 2007), 1–33. <https://doi.org/10.1029/2005RG000183>.
- [FYZ\*18] FENG Y., YOU H., ZHANG Z., JI R., GAO Y.: Hypergraph neural networks. *AAAI 2019*, (2018).
- [Gab77] GABOW H. N.: Two algorithms for generating weighted spanning trees in order. *SIAM Journal on Computing* 6, 1 (1977), 139–150.
- [GDG\*17] GUÉRIN É., DIGNE J., GALIN E., PEYTAIE A., WOLF C., BENES B., MARTINEZ B.: Interactive example-based terrain authoring with conditional generative adversarial networks. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 1–13.
- [GGG\*13] GÉNEVAUX J.-D., GALIN E., GUÉRIN E., PEYTAIE A., BENES B.: Terrain generation using procedural models based on hydrology. *ACM Transactions on Graphics* 32, 4 (July 2013), 1–13. <https://doi.org/10.1145/2461912.2461996>.
- [GGP\*19] GALIN E., GUÉRIN E., PEYTAIE A., CORDONNIER G., CANI M.-P., BENES B., GAIN J.: A review of digital terrain modeling. *Computer Graphics Forum* 38, 2 (2019), 553–577. <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13657>.
- [GPAM\*20] GOODFELLOW I., POUGET-ABADIE J., MIRZA M., XU B., WARDE-FARLEY D., OZAIR S., COURVILLE A., BENGIO Y.: Generative adversarial networks. *Communications of the ACM* 63, 11 (2020), 139–144.
- [GPM\*22] GUÉRIN E., PEYTAIE A., MASNOU S., DIGNE J., SAUVAGE B., GAIN J., GALIN E.: Gradient terrain authoring. *Computer Graphics Forum* 41, (2022), 85–95.
- [Gus97] GUSFIELD D.: Algorithms on stings, trees, and sequences: Computer science and computational biology. *ACM Sigact News* 28, 4 (1997), 41–60.

- [HAYC20] HUANG Z., ARIAN A., YUAN Y., CHIU Y.-C.: Using conditional generative adversarial nets and heat maps with simulation-accelerated training to predict the spatiotemporal impacts of highway incidents. *Transportation Research Record* 2674, 8 (2020), 836–849.
- [HCX\*22] HE K., CHEN X., XIE S., LI Y., DOLLÁR P., GIRSHICK R.: Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (New Orleans, LA, 2022), IEEE, pp. 16000–16009.
- [HHM\*23] HU Z., HU K., MO C., PAN L., WANG Z.: Terrain diffusion network: Climatic-aware terrain generation with geological sketch guidance. *arXiv preprint arXiv:2308.16725* (2023).
- [Hor45] HORTON R. E.: Erosional development of streams and their drainage basins: Hydrophysical approach to quantitative morphology. *Geological Society of America Bulletin* 56, 3 (1945), 275–370.
- [HRU\*17] HEUSEL M., RAMSAUER H., UNTERTHINER T., NESSLER B., HOCHREITER S.: GANs trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in Neural Information Processing Systems* 30, (2017), 6629–6640.
- [HZ10] HORÉ A., ZIOU D.: Image quality metrics: PSNR vs. SSIM. In *2010 20th International Conference on Pattern Recognition* (Istanbul, Turkey, 2010), IEEE, pp. 2366–2369. <https://doi.org/10.1109/ICPR.2010.579>.
- [IZZE17] ISOLA P., ZHU J.-Y., ZHOU T., EFROS A. A.: Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu, HI: 2017), IEEE, pp. 1125–1134.
- [JD88] JENSON S. K., DOMINGUE J. O.: Extracting topographic structure from digital elevation data for geographic information system analysis. *Photogrammetric Engineering and Remote Sensing* 54, 11 (1988), 1593–1600.
- [JFBB10] JONES M. D., FARLEY M., BUTLER J., BEARDALL M.: Directable weathering of concave rock using curvature estimation. *IEEE Transactions on Visualization and Computer Graphics* 16, 1 (January 2010), 81–94. <https://doi.org/10.1109/TVCG.2009.39>.
- [JGP17] JANG E., GU S., POOLE B.: Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations* (Toulon, France, 2017), OpenReview.net.
- [KBKŠ09] KRIŠTOF P., BENES B., KŘIVÁNEK J., ŠŤAVA O.: Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum (Proceedings of Eurographics 2009)* 28, 2 (March 2009), 469–478. <http://www2.tech.purdue.edu/cgt/Facstaff/bbenes/private/papers/EG09SPH.zip>.
- [KDRB21] KIM H., DISCHLER J.-M., RUSHMEIER H., BENES B.: Edge-based procedural textures. *The Visual Computer* 37, 9 (2021), 2289–2302. <https://doi.org/10.1007/s00371-021-02212-4>.
- [KHM\*20] KRS V., HAEDRICH T., MICHELS D. L., DEUSSEN O., PIRK S., BENES B.: Wind Erosion: Shape modifications by interactive particle-based erosion and deposition. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation — Posters* (2020), Michels D. L., (Ed.), The Eurographics Association. <https://doi.org/10.2312/sca.20201216>.
- [KMN88] KELLEY A. D., MALIN M. C., NIELSON G. M.: Terrain simulation using a model of stream erosion. *ACM Transactions on Graphics* (1988), 263–268. <http://doi.acm.org/10.1145/54852.378519>.
- [KW17] KIPF T. N., WELLING M.: Semi-supervised classification with graph convolutional networks. In *The International Conference on Learning Representations (ICLR)* (Toulon, France, 2017), OpenReview.net.
- [LGP\*23] LOCHNER J., GAIN J., PERCHE S., PEYTAVIE A., GALIN E., GUÉRIN E.: Interactive authoring of terrain using diffusion models. *Computer Graphics Forum* 42, 7 (2023), e14941.
- [LM17] LI P., MILENKOVIC O.: Inhomogeneous hypergraph clustering with applications. *Advances in Neural Information Processing Systems* 30 (2017), 1237–1246.
- [LM18] LI P., MILENKOVIC O.: Submodular hypergraphs: p-Laplacians, Cheeger inequalities and spectral clustering. In *International Conference on Machine Learning* (Stockholm, Sweden, 2018), PMLR, pp. 3014–3023.
- [Man83] MANDELBROT B. B.: *The Fractal Geometry of Nature*. W.H. Freeman and Company, San Francisco, 1983.
- [MKM89] MUSGRAVE F. K., KOLB C. E., MACE R. S.: The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH '89: Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1989), ACM Press, pp. 41–50. <https://doi.acm.org/10.1145/74333.74337>.
- [PFS\*19] PARK J. J., FLORENCE P., STRAUB J., NEWCOMBE R., LOVEGROVE S.: DeepSDF: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Long Beach, CA, June 2019), IEEE.
- [PPB\*23] PERCHE S., PEYTAVIE A., BENES B., GALIN E., GUÉRIN E.: Authoring terrains with spatialised style. *Computer Graphics Forum* 42, 7 (October 2023), e14936. <https://doi.org/10.1111/cgf.14936>.
- [RBL\*22] ROMBACH R., BLATTMANN A., LORENZ D., ESSER P., OMMER B.: High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (New Orleans, LA, June 2022), IEEE, pp. 10684–10695.
- [RKČ\*22] RAJASEKARAN S. D., KANG H., ČADÍK M., GALIN E., GUÉRIN E., PEYTAVIE A., SLAVÍK P., BENES B.: PTRM: Perceived

- terrain realism metric. *ACM Transactions on Applied Perception* 19, 2 (July 2022), 1–22. <https://doi.org/10.1145/3514244>.
- [RSDM19] ROTT SHAHAM T., DEKEL T., MICHAELI T.: Singan: Learning a generative model from a single natural image. In *Computer Vision (ICCV), IEEE International Conference on* (2019), IEEE, 4510–4519.
- [SBBK08] STAVA O., BENES B., BRISBIN M., KRIVANEK J.: Interactive terrain modeling using hydraulic erosion. *Eurographics/Siggraph Symposium on Computer Animations SCA 27*, (2008), 201–210. <http://www2.tech.purdue.edu/cgt/Facstaff/bbenes/private/papers/Stava08SCA.zip>, <https://doi.org/10.2312/SCA/SCA08/201-210>.
- [VCC\*18] VELIČKOVIĆ P., CUCURULL G., CASANOVA A., ROMERO A., LIO P., BENGIO Y.: Graph attention networks. In *International Conference on Learning Representations* (2018).
- [vdMH08] VAN DER MAATEN L., HINTON G.: Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605. <http://jmlr.org/papers/v9/vandermaaten08a.html>.
- [WBSS04] WANG Z., BOVIK A., SHEIKH H., SIMONCELLI E.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.
- [WCMT07] WOJTAN C., CARLSON M., MUCHA P. J., TURK G.: Animating corrosion and erosion. In *Proceedings of the Eurographics Workshop on Natural Phenomena, NPH 2007*, (Prague, Czech Republic, 2007), Eurographics Association, pp. 15–22. <https://doi.org/10.2312/NPH/NPH07/015-022>.
- [WYL\*22] WANG P., YANG S., LIU Y., WANG Z., LI P.: Equivariant hypergraph diffusion neural operators. *arXiv preprint arXiv:2207.06680* (2022).
- [XHLJ19] XU K., HU W., LESKOVEC J., JEGELKA S.: How powerful are graph neural networks? In *The International Conference on Learning Representations (ICLR)* (New Orleans, LA, USA, 2019), OpenReview.net.
- [YCC\*24] YANG Z., CORDONNIER G., CANI M.-P., PERRENOUD C., BENES B.: Unerosion: Simulating terrain evolution back in time. *Computer Graphics Forum* 43, (2024), 1–15. <https://doi.org/10.1111/cgf.15182>.
- [YNGS\*08] YU A., NOVO-GRADAC A., SHAW G., UNGER G., RAMAS-IZQUIERDO L., LUKEMIRE A.: The lunar orbiter laser altimeter (LOLA) laser transmitter [6871-109]. In *Proceedings-SPIE the International Society for Optical Engineering* (2008), vol. 6871, International Society for Optical Engineering; 1999, p. 6871.
- [ZLB\*19] ZHAO Y., LIU H., BOROVNIKOV I., BEIRAMI A., SANJABI M., ZAMAN K.: Multi-theme generative adversarial terrain amplification. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–14.
- [ZLZ\*18] ZHANG Z., LIN H., ZHAO X., JI R., GAO Y.: Inductive multi-hypergraph learning and its application on view-based 3D object classification. *IEEE Transactions on Image Processing* 27, 12 (2018), 5957–5968.
- [ZPIE17] ZHU J.-Y., PARK T., ISOLA P., EFROS A. A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision* (Venice, Italy, 2017), IEEE, pp. 2223–2232.
- [ZRA23] ZHANG L., RAO A., AGRAWALA M.: Adding conditional control to text-to-image diffusion models. In *IEEE International Conference on Computer Vision (ICCV)* (Vancouver, Canada, 2023), IEEE.
- [ZSTR07] ZHOU H., SUN J., TURK G., REHG J. M.: Terrain synthesis from digital elevation models. *IEEE transactions on visualization and computer graphics* 13, 4 (2007), 834–848.

## Appendix A

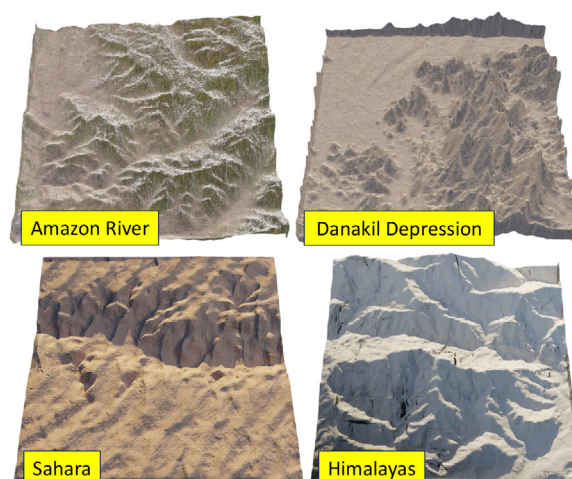
### A.1. Additional results

We show additional terrain in Figure A1.

### A.2. Large sketches and regions

We preprocess the region of the sketch due to the limitations of the GPU memory. The preprocessing focuses on constructing the region  $r$  with corresponding pixels  $p$ . The region is affected by a sketch (Section 3). However, the terrain is too informative to make the entire sketch into one embedding.

We split the long sketch into small sketches. Note that this operation will not affect the topology information since Equation (1) injects partial sketch information into nearby pixels and then updates the nearby sketches. The original sketches are still connected. Inspired by the MAE [HCX\*22], 5% small sketches will be masked in each iteration. This can help the model fully explore the data and



**Figure A1:** The ground truth of the Amazon River, the Sahara, the Danakil Depression, the Himalayas.

**Algorithm A1.** Rotation and Normalization.

---

**Input :** A  $n \times 2$  array  $P : \{(P_x^i, P_y^i)\}, i = 1, \dots, n,$   
 A  $m \times 2$  array  $Q : \{(Q_x^j, Q_y^j)\}, j = 1, \dots, m.$   
**Output:** Normalized  $P'$ ,  
 Rotated normalized  $Q'$

```

1 // Define the new direction
2 if  $P_z^0 > P_z^n$  then
3   Reverse (P)
4 end if
5 if  $Q_z^0 > Q_z^m$  then
6   Reverse (Q)
7 end if
8  $v_1 = \langle P_x^n - P_x^1, P_y^n - P_y^1 \rangle$ 
9  $v_2 = \langle Q_x^m - Q_x^1, Q_y^m - Q_y^1 \rangle$ 
10  $\theta = \text{Calculate\_Angle\_from\_}v_1\text{\_to\_}v_2(v_1, v_2)$ 
11  $\theta' = \text{Round}(\theta/\pi) \times \frac{\pi}{8}$ 
12 // Normalization
13  $P'[0 : n - 1, :] = P[1 : n, :] - P[0 : n - 1, :]$ 
14  $P'_{sum}[:] = \sqrt{P'[:, 0]^2 + P'[:, 1]^2}$ 
15  $P' = P'/P'_{sum}$  // Element-wise divide
16  $Q'[0 : m - 1, :] = Q[1 : m, :] - Q[0 : m - 1, :]$ 
17  $Q'_{sum}[:] = \sqrt{Q'[:, 0]^2 + Q'[:, 1]^2}$ 
18  $Q' = Q'/Q'_{sum}$  // Element-wise divide
19 // Rotate the normalized  $Q'$ 
20  $Q' = \text{Rotate\_Angle}(Q', -\theta')$  return  $P', Q'$ 
```

---

**Algorithm A2.** Sketch Alignment.

---

**Input :** A  $n \times 3$  array  $P : \{(P_x^i, P_y^i, P_z^i)\}, i = 1, \dots, n,$   
 $k \times 3$  arrays  $Q_k : \{(Q_x^j, Q_y^j, Q_z^j)\}, j = 1, \dots, m.$   
**Output:** Index  $i$  of the aligned sketch.

```

1 // Initialization
2  $L[1..k] \leftarrow 0$ 
3 for  $i \leftarrow 1$  to  $k$  do
4   // Rotate and Normalize two vectors
5    $P', Q_i = \text{Rotation\_and\_Normalization}(P, Q_i)$ 
6   // return the index of longest common
   substring
7    $I_{P'}, I_{Q_i} = \text{Longest\_Common\_Substring}(P', Q_i)$ 
8    $L[i] = \text{length}(I_{P'})$ 
9 end for
10 Choose the top 10 largest values in  $L$ .
11 Compute height differences on the LCS  $I_{P'}$  and  $I_{Q_i}$ 
12 Randomly choose one sketch from the top 5 smallest height
   differences.
13 return Sketch index  $i$  with the smallest height difference
```

---

the correlation among sketches while avoiding the over-fitting problem.

### A.3. Initialization of $f_p^{(t)}$ and $E_r^{(t)}$

Pixels are assigned to some initial values depending on whether they consist of a sketch in Equation (A1). A nonlinear multiple layer perceptron (MLP)  $MLP_0$  is applied to increase the dimension for the

**Table B1:** The numerical result of the training process.

Terrain	Alps	Grand Canyon	Moon
$l_{gr} (\times 10^{-4})$	7.50	8.38	5.88
$l_{gradient} (\times 10^{-4})$	1.70	2.25	1.53
$l_{sketch} (\times 1)$	0.37	0.08	0.26
$l_{smooth} (\times 1)$	0.01	0.05	0.05
$l_{further} (\times 10^{-4})$	0.37	2.45	8.88
$l_{ssim}$	0.97	0.97	0.99

Note: The first column is our loss functions, and the next three columns show the number of steps when we stopped training for different terrains.

later process, and  $E_r^{(t)}$  is randomly sampled from normal distribution  $\mathcal{N}(0, 1)$ :

$$f_p^0 = \begin{cases} MLP_0(1) & p \in \text{ridge} \\ MLP_0(-1) & p \in \text{valley.} \\ MLP_0(0) & \text{others} \end{cases} \quad (\text{A1})$$

### A.4. Design of $e_p^r$

Edge information  $e_p^r$  depicts the relation between a pixel  $p$  and a region  $r$ . A representative yet generalizable feature is essential. We propose the below items for the  $e_p^r$ :

- The relative position: the index  $proj_p^r$  of projection  $p$  on sketch  $e$  and the distance from  $p$  to the projection  $dist_p^r$ .
- Sketch indicators  $I_{sketch}$ , a 2D vector: the first dimension indicates whether the pixel is on the ridge; the second indicates the valley.
- embedding  $Emb(\cdot)$  of sketches index  $Idx_{sketch}$ . Each sketch has a unique embedding for a specific local feature.
- Gaussian noise  $n$ , where  $n \sim \mathcal{N}(0, 1)$  for randomness.

### A.5. Choice of $Enc_1, Enc_2, Gath_r$ and $Gath_p$

For  $Enc_1$  and  $Enc_2$ , we simply apply an MLP. However, to have a better representation power, the summation of  $f_p^{(t)}$  and  $E_r^{(t)}$  is applied. This is because it enables the set function to be both permutation invariant and equivariant [WYL\*22]:

$$Enc_1(f_p^t, e_p^r) = MLP_1 \left( f_p^t, e_p^r, \sum_{\forall p, p \in r} f_p^{(t)} \right)$$

$$Enc_2(E_r^{(t)}, e_p^r) = MLP_2 \left( E_r^{(t)}, e_p^r, \sum_{\forall r, p \in r} E_r^{(t)} \right).$$

The implementation is adopted from [WYL\*22]. However, in our work, since all information  $e_p^r$  is stored on edge, we concatenate the  $e_p^r$  to the node information  $f_p^{(t)}$  and  $E_r^{(t)}$ .

For  $Gath_r$  and  $Gath_p$ , we choose the graph attention model [VCC\*18]. Again, we inject  $e_p^r$  to each node to incorporate edge information.



## Appendix B: More Experiments

**Expressive Power** is the ability to overfit the model. We show the training loss for three different terrains. The  $l_{gt}$  and other losses are shown in Table B1.

Table B1 shows that all the datasets have a similar performance of  $l_{gt}$  and  $l_{gradient}$  around  $7.25 \times 10^{-4}$  and  $1.83 \times 10^{-4}$ , indicates that

our model can reconstruct all these terrains while keeping the high-frequency features. The  $l_{sketch}$  and  $l_{smooth}$  values are small, ensuring that each sketch is smooth and follows the common knowledge of the shape of the ridge and valley. The  $l_{further}$  is around  $3.90 \times 10^{-4}$ . This forces the influence of a sketch to decrease to zero when the distance is far enough. The  $l_{sim}$  around 0.98 shows they are almost identical.