Virtual World Creation

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering by Research

by

Aryamaan Jain
2019121002
aryamaan.jain@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500 032, INDIA
July 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Virtual World Creation" by Aryamaan Jain, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. Avinash Sharma
Date	Co-Adviser: Prof. K S Rajan

To my parents

Acknowledgments

I would like to express my deepest gratitude to the following individuals without whom this thesis would not have been possible:

First and foremost, I extend my sincere thanks to my advisor, Dr Avinash Sharma, for his invaluable guidance, support, and encouragement throughout my research journey. His vast knowledge, insightful suggestions, and constructive criticism have been instrumental in shaping my research work and guiding me towards the successful completion of this thesis. I am also deeply grateful to Dr K S Rajan, my co-advisor, for his expert advice and domain knowledge in GIS, which proved to be of great help in the development of my research. His encouragement and support have been invaluable throughout this journey.

I extend my heartfelt appreciation to Shanthika Naik for her invaluable contributions to the terrain generation research work. Her expertise and dedication were instrumental in the success of the project. I also express my sincere gratitude to Jyoti Sunkara and Ishaan Shah, who collaborated with me on tree generation. Their insightful comments, valuable suggestions, and technical assistance were crucial to the successful completion of the paper. I would like to acknowledge Saravanan Senthil for his invaluable insights into large-scale terrain generation and tree population during his project work. His contributions have deepened my understanding of the challenges involved in generating and populating terrain at scale. Additionally, I am grateful to Ashish Kubade and Diptiben Patel for their guidance and support during the initial stages of my research on terrain super-resolution.

Finally, I would like to thank my parents for their unwavering support and encouragement throughout my academic journey. Their love and support have been a constant source of inspiration to me, and I could not have achieved my goals without them.

Abstract

Synthesis, capture and analysis of a highly complex 3D terrain structure are essential for critical applications such as river/flood modelling, disaster mitigation planning, landslide modelling and flight simulation. On the other hand, synthesis of natural-looking 3D terrains finds its applications in the entertainment industry such as computer gaming and VFX. This thesis explores novel learning-based techniques for the generation of immersive and realistic 3D virtual environments, catering to the needs of the aforementioned applications. The generation of virtual worlds involves multiple components, including terrain, vegetation, and other objects. We primarily focus on three key aspects for virtual world generation: 1) develop novel AI-enabled 3D terrain authoring solutions based on real-world satellite and aerial data using a learning-based framework, 2) L-systems grammar-based 3D tree generation and 3) rendering techniques that are used to create high-quality visualizations of the generated world.

Terrain generation is a critical component of 3D virtual world generation, as it provides the foundational structure for the environment. Traditional techniques for 3D terrain generation involve procedural generation, which relies on mathematical algorithms to generate landscapes. However, deep learning techniques have shown promise in generating more realistic terrain, as they can learn from real-world data to produce new, varied, and realistic landscapes. In this thesis, we explore the use of deep learning techniques for 3D terrain generation, which can produce realistic and varied terrains with high visual fidelity. Specifically, we propose two learning-based novel frameworks for *Interactive 3D Terrain Authoring & Manipulation* and *Adaptive Multi-Resolution Infinite Terrain Generation*. In addition to terrain generation, vegetation is another important component of virtual world generation. Trees and other plants provide visual interest and can help create a more immersive environment. L-systems are a popular technique for generating realistic vegetation, as they are capable of generating complex structures that resemble real-world plants. In this thesis, we propose a variant of the L-systems for 3D tree generation and compare the results to traditional procedural generation techniques. Finally, rendering is a critical component of 3D virtual world generation, as it is responsible for creating the final visual output that users will see. In addition to terrain and tree generation, this thesis also covers rendering techniques used to visualize the generated virtual world. We explore the use of real-time rendering techniques in conjunction with terrain generation to achieve high-quality visual results while maintaining performance.

Overall, the research presented in this thesis aims to advance the state-of-the-art in virtual world generation and contribute to the development of more realistic and immersive virtual environments. We performed extensive empirical evaluation on publicly available datasets to report details qualitative and quantitative results demonstrating the superiority of the proposed methods over existing solutions in the literature.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Virtual world creation has been a topic of research for several decades, with significant advancements in technology leading to more realistic and immersive environments as shown in Figure 1.1. The synthesis, capture, and analysis of intricate 3D terrain structures play a crucial role in various vital endeavors. These include river/flood modeling, disaster mitigation planning, landslide modeling, and flight simulation. Simultaneously, the creation of realistic-looking 3D terrains finds its practical utility in the entertainment sector, specifically in computer gaming and VFX.

In this thesis, we explore novel learning-based techniques for the generation of immersive and realistic 3D virtual environments, catering to the needs of the aforementioned applications. The generation of virtual worlds involves multiple components, including terrain, vegetation, and other objects. Our primary focus is on two key aspects of virtual world generation: 3D terrain authoring solutions based on real-world satellite and aerial data using a learning-based framework and L-systems grammar-based 3D tree generation. Additionally, we also discuss rendering techniques used to create high-quality visualizations of the generated world.

Terrain generation is a critical component of 3D virtual world generation, as it provides the foundational structure for the environment. Traditional techniques [75] for 3D terrain generation involve procedural generation, which relies on mathematical algorithms to generate landscapes. However, deep learning techniques have shown promise in generating more realistic terrain, as they can learn from real-world data to produce new, varied, and realistic landscapes. This thesis delves into the application of deep learning techniques in the field of 3D terrain generation. These techniques enable the creation of diverse terrains. Our research focuses on two innovative frameworks: Interactive 3D Terrain Authoring & Manipulation and Adaptive Multi-Resolution Infinite Terrain Generation.

In addition to terrain generation, vegetation is another important component of virtual world generation. Trees and other plants provide visual interest and can help create a more immersive environment. L-systems are a popular technique for generating realistic vegetation, as they are capable of generating complex structures that resemble real-world plants. This thesis introduces a modified version of L-systems designed for generating 3D trees, and evaluates its performance by comparing it to conventional procedural generation methods.

Finally, rendering is a critical component of 3D virtual world generation, as it is responsible for creating the final visual output that users will see. In addition to terrain and tree generation, this thesis also covers rendering techniques used to visualize the generated virtual world. We explore the use of real-time rendering techniques in conjunction with terrain generation to achieve high-quality visual results while maintaining performance.

The main objective of this thesis is to advance the state-of-the-art in virtual world generation and contribute to the development of more realistic and immersive virtual environments. We performed extensive empirical evaluation on publicly available datasets to report details qualitative and quantitative results demonstrating the superiority of the proposed methods over existing solutions in the literature. In the next chapters, we discuss the details of our proposed frameworks for 3D terrain authoring, 3D tree generation, and rendering techniques used to create high-quality visualizations of the generated virtual world.

Figure 1.1: A view of southwest US in VBS4 [6] featuring a vast virtual environment with terrain and trees.

1.1 Motivation

Virtual world generation has become an increasingly important area of research in computer vision, computer graphics, mixed reality and simulation. Virtual worlds are used in various applications such as entertainment, education, and training, and they provide a platform for exploring and experiencing different environments. However, creating realistic virtual environments is a challenging task that re-

quires expertise in multiple domains, including computer graphics, vision, simulation, GIS and artificial intelligence.

One of the critical aspects of creating realistic virtual environments is generating the terrain and vegetation that make up the environment. Traditional methods for generating terrain and vegetation often require significant manual effort and expertise, as well as complex algorithms and techniques. Recent advances in deep learning have made it possible to generate realistic and diverse terrain and vegetation along with opening possibilities for incorporating rendering algorithms seamlessly.

The potential impact of our work is significant. By reducing the time and effort required to create virtual environments, we can lower the barrier to entry for creating immersive and realistic virtual worlds. This, in turn, can enable a wider range of applications for virtual environments. Furthermore, our work can also have applications in game development, where virtual environments are critical for creating immersive gameplay experiences. By generating diverse and realistic virtual environments, we can enable game developers to create more engaging and captivating worlds that keep players coming back for more.

In this thesis, we will focus on the development of algorithms for terrain and tree generation in virtual worlds, while also ensuring efficient rendering. Our objective is to create virtual environments that realistically mimic real-world landscapes and offer an immersive and engaging user experience. We will investigate state-of-the-art techniques for terrain and tree generation, examine effective methods for rendering them, and propose novel algorithms to overcome the challenges faced in this area. Furthermore, we will evaluate the effectiveness of our algorithms using user studies and compare our findings with existing techniques. Ultimately, this thesis will advance the creation of virtual worlds and have diverse applications across various fields.

1.2 Problem Statement

Synthesis, capture and analysis of a highly complex terrain structure (as shown in Figure 1.2) is essential for critical applications such as river/flood modeling, disaster mitigation planning, landslide modeling and flight simulation [17]. These applications demand high resolution details on terrain. Terrain are commonly represented as Digital Elevation Models (DEM), which is a digital file that contains elevation data for each point on the Earth's terrain and employed in GIS, cartography, engineering, and environmental studies. DEMs extracted from remotely sensed data are available at coarser resolution for majority of the planet while only finer resolution is available for limited areas due to challenges in acquisition and cost implications (e.g., LiDAR and SfM/Stereo).

On the other hand, synthesis of a natural looking terrain as Digital Surface Model (DSM) finds its applications in entertainment industry such as computer gaming, science fiction and fantasy genre cinematography. The DSMs synthesized with simulation based models include high resolution DEMs and vegetation layer [5] on top of it to enhance the realism of natural scenes and to further augment

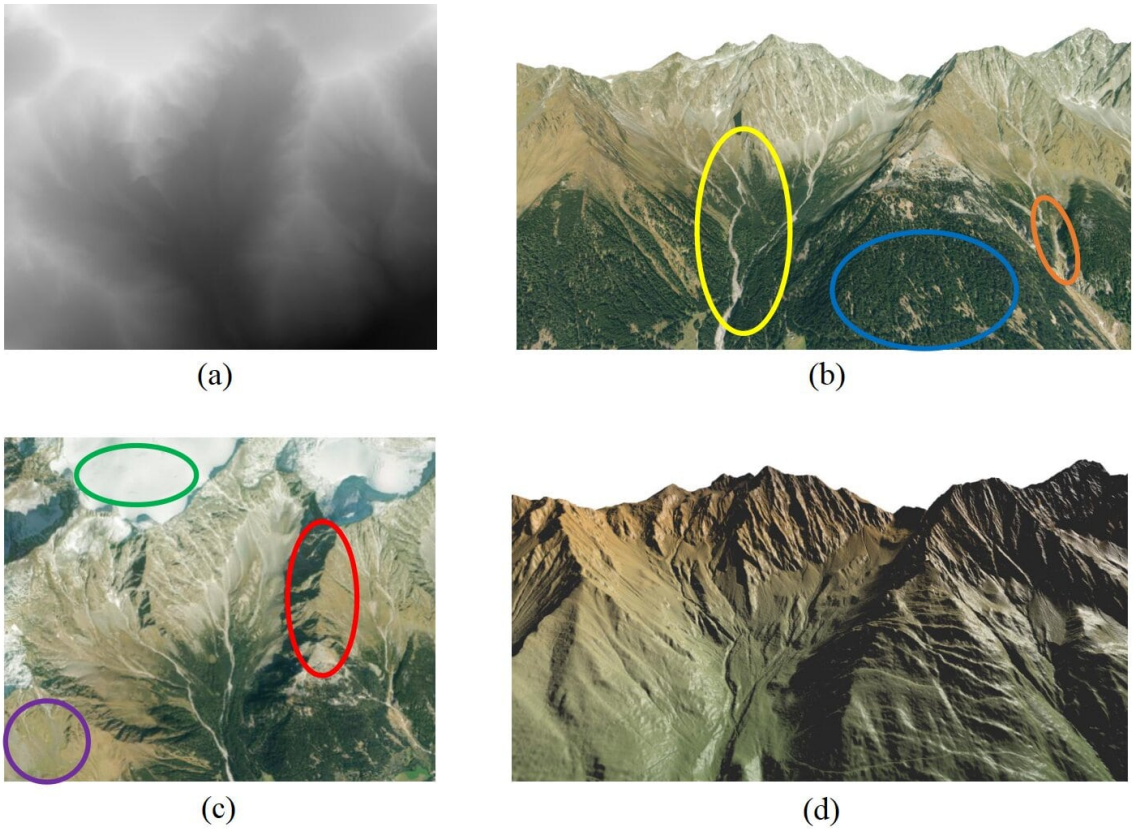


Figure 1.2: Terrain surface represented as (a) Digital Elevation Model, (b), (c) Aerial Image, (d) Hill shade Model. It consists of complex structural information such as high mountainous region, dense vegetation, heavy snowfall, complex river networks, deep valleys and bare land.

the probes such as trees, plants, grass, etc. However, these DSMs often lack realism as the simulation models typically do not learn from real world terrain data.

We aim to develop novel AI-enabled terrain authoring solutions with high degree of user control based on real-world satellite and aerial data using a learning based framework. This would enable generation of high resolution DEMs and respective vegetation layers. The literature in this direction is fairly at nascent stage where the state-of-the-art methods either enhances the resolution details of the captured low-resolution DEMs using multiple modalities such as aerial images [3, 45, 46] or synthesize the natural looking terrain [23, 4] with limited user control. However, the existing methods limits the user control and lacks the realism of natural looking terrain when accessed at different scales.

Some of our goals are (1) Using sketches and level-set as inputs, generate realistic looking low resolution DEM as illustrated in Figure 1.3. Enhance the resolution of a DEM in order to add low level details and get high resolution DEM. (2) Add user control mechanism to edit the existing sketch and level sets to artistically change the DEMs for user specific requirements maintaining realism in modified

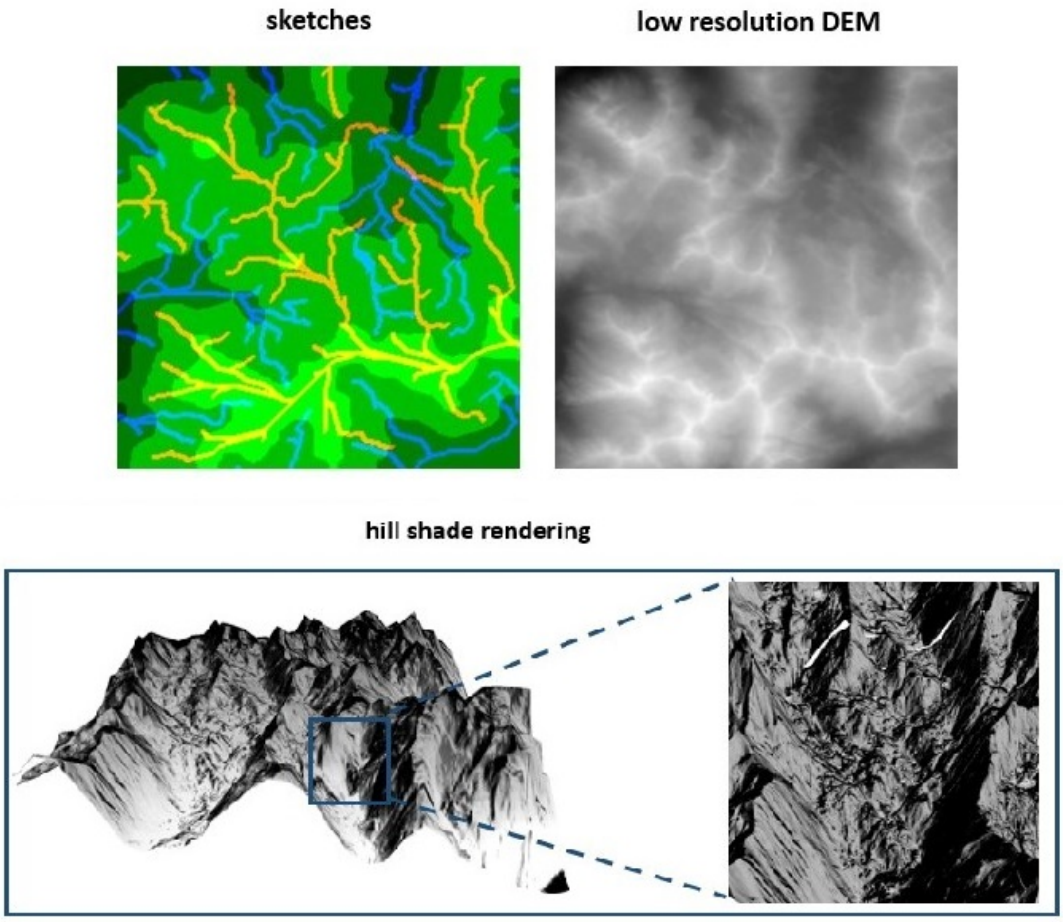


Figure 1.3: Converting user drawn sketch to terrain gives high degree of control to the user.

DEM. (3) Generate a rich library which contains diverse 3D tree and grass models using real-world data of 2D image samples. Render the vegetation layer using the library and integrate it with high resolution DEM to get fully synthesized digital surface model. (4) Rendering of thousands of trees on a large terrain surface is a computationally challenging for a real-time navigation applications. We would like to explore the multi-resolution representation of rendering details. A high level overview of the problem statement is shown in Figure 1.4.

1.3 Application areas

Virtual worlds have diverse applications ranging from gaming and simulations to education and tourism. Below, we will explore some of these applications.

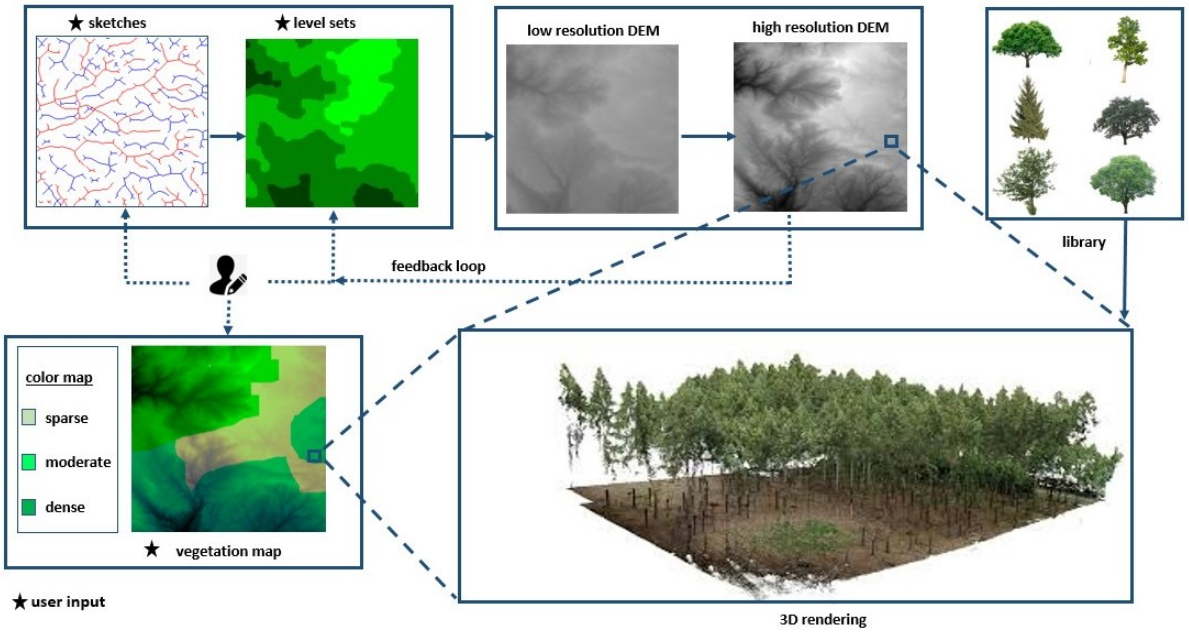


Figure 1.4: A high level overview of the problem statement.

1. **Gaming**: Virtual worlds have become a popular medium for entertainment and gaming (Figure 1.5), providing players with immersive experiences through realistic terrain and vegetation. Game developers leverage virtual world creation techniques to generate diverse environments that align with the game's theme and storyline. For instance, they can design lifelike landscapes for survival games or craft fantasy worlds for role-playing games. A notable example is Minecraft, which employs a terrain generation algorithm that combines various noise functions to produce a range of biomes, landscapes, and natural features, including mountains, caves, and rivers. This approach yields a broad range of environments for players to explore and interact with. Additionally, Minecraft's tree generation algorithms use specific rules based on the type of biome, tree, and availability of light and space to randomly generate trees in various biomes.

2. **Films**: Animated films benefit greatly from virtual environments (Figure 1.6) that offer immersive settings for characters to engage with. Using terrain and tree generation techniques, creators can design highly realistic and detailed backgrounds that provide a natural and believable environment for characters to inhabit. This results in an enhanced viewing experience where the virtual world feels seamless and integrated with the story.

3. **Flight simulation**: Terrain generation is a critical component of flight simulations, as it provides a realistic and immersive experience for pilots and trainees as shown in Figure 1.7. One popular application of terrain generation in flight simulations is the creation of virtual environments and rendering them in real-time that simulate real-world conditions, allowing pilots to train in different scenarios and environments.

(a) Minecraft [64]

(b) Elder Scrolls V: Skyrim [87]

Figure 1.5: Terrain and trees in games.

Figure 1.6: A shot from the film *The Lion King* [41] with terrain and vegetation in the background

4. **Emergency Response Training**: Virtual worlds are also used in emergency response training to simulate realistic scenarios without endangering the lives of emergency responders. Terrain and tree generation techniques can help create realistic landscapes that replicate real-world environments, providing trainees with an immersive and effective learning experience.

5. **Architecture and Landscape Design**: Virtual worlds can be used in architecture and landscape design to simulate different building and landscape designs. Terrain and tree generation techniques can help create realistic models of the environment, allowing architects and designers to visualize the impact of their designs on the surrounding environment like Figure 1.8. This can aid in sustainable design practices and in creating buildings and landscapes that are in harmony with the natural environment.

6. **Education**: Virtual worlds are increasingly being used in education to provide interactive and engaging learning experiences. Terrain and tree generation techniques can help create educa-

Figure 1.7: Vast terrain and detailed trees in *Microsoft Flight Simulator* [63].

tional virtual environments that simulate different ecosystems, enabling students to learn about geography, geology, and ecology in an immersive and interactive way.

7. **Tourism**: Virtual worlds can be used in tourism to provide a preview of potential travel destinations. Terrain and tree generation techniques can help create realistic virtual environments that allow tourists to explore and experience a destination before they actually travel there. For example, virtual reality tours of natural landmarks and tourist destinations can be created using virtual world creation techniques.

8. **Environmental Studies and Conservation**: Virtual worlds can be used to simulate different ecosystems and their dynamics, enabling researchers to study environmental changes and their impact on flora and fauna. Terrain and tree generation techniques can help create realistic models of ecosystems, allowing researchers to study the impact of climate change, natural disasters, and human activities on different environments. This can aid in conservation efforts and help in creating sustainable solutions.

9. **Art and Design**: Virtual worlds can be used in art and design to create virtual sculptures, installations, and other artworks that are inspired by natural landscapes. Terrain and tree generation techniques can help create realistic models of different environments, allowing artists to create immersive and interactive artworks that replicate natural landscapes. This can create new opportunities for artists to explore and experiment with different forms of art and design.

10. **Real Estate and Property Development**: Virtual worlds can be used in real estate and property development to simulate different types of properties and their impact on the environment. Ter-

Figure 1.8: Virtual trees covering the *Central Vista Project* design [11].

rain and tree generation techniques can help create realistic models of the environment, allowing property developers to visualize the impact of their developments on the surrounding landscape. This can aid in sustainable property development practices and in creating properties that are in harmony with the natural environment.

Despite the progress that has been made in virtual world creation, there are still challenges that need to be addressed. For example, generating realistic vegetation can be computationally expensive, and there is a need for more efficient algorithms that can generate vegetation in real-time. Additionally, the ability to generate dynamic and interactive landscapes that respond to user input and environmental factors is an area of active research.

Overall, our thesis aims to contribute to the growing field of virtual world generation using deep learning techniques, with the potential to impact various applications and fields. We believe that our work can help advance the state-of-the-art in virtual world generation, enabling designers and developers to create more immersive and realistic virtual environments.

1.4 Research Landscape

1.4.1 Computer Vision Landscape

Computer vision techniques such as image generation and super-resolution can be used in the creation of virtual worlds, particularly for generating realistic terrain. For example, image generation

techniques can be used to synthesize new terrain height-maps and features based on real-world examples or other references (chapter 2, chapter 3, chapter 5), while super-resolution techniques can be used to enhance the resolution and detail of existing terrain data (chapter 3, chapter 5). These techniques can help create more immersive and visually appealing virtual worlds for various applications, such as gaming, simulation, and training.

1.4.1.1 Generation

Generative modeling is a subfield of machine learning that focuses on building models that can generate new data samples that are similar to the data that they were trained on. Generative models are particularly useful in a variety of applications such as terrain generation, terrain completion (inpainting/outpainting) and terrain super-resolution.

Three popular generative models are Generative Adversarial Networks (GANs) [21] (chapter 2, chapter 5), Variational Autoencoders (VAEs) [44] (chapter 2), and Diffusion Models [69, 13] (chapter 3). Each of these models has its own strengths and weaknesses, and understanding the differences between them is important for choosing the right model for a particular application.

Generative Adversarial Networks (GANs) are a type of generative model that involves training two neural networks: a generator network and a discriminator network. The generator network takes as input a random noise vector and generates a sample that is intended to look like it came from the training data. The discriminator network takes as input a sample, and tries to classify whether it is a real training sample or a fake sample generated by the generator network. The two networks are trained simultaneously, with the generator network trying to fool the discriminator network, and the discriminator network trying to correctly classify the samples. The training process can be represented by the following objective function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{1.1}$$

where G is the generator network, D is the discriminator network, x is a sample from the training data distribution p_{data}, z is a noise vector sampled from a prior distribution p_z, and $G(z)$ is the output of the generator network when given input z.

Variational Autoencoders (VAEs) are another popular type of generative model that involves training an encoder network and a decoder network. The encoder network takes as input a sample from the training data, and outputs a distribution over a lower-dimensional latent space. The decoder network takes a sample from the latent space and generates a sample that is intended to look like it came from the training data. The training process involves maximizing a lower bound on the log-likelihood of the training data, and can be represented by the following objective function:

$$\mathcal{L}_{VAE} = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + KL(q_\phi(z|x)||p(z)) \tag{1.2}$$

where $q_\phi(z|x)$ is the encoder network, $p_\theta(x|z)$ is the decoder network, z is a latent variable sampled from a prior distribution $p(z)$, x is a sample from the training data distribution, and KL denotes the Kullback-Leibler divergence between two distributions.

The diffusion models are another class of generative models introduced in the seminal work by [90]. These models were further improved by [30, 69, 13] until recently when they eventually beat GANs on image synthesis task. They involve modeling the process of diffusion, or the gradual smoothing out of sharp features in a signal. The diffusion process is modeled using a sequence of stochastic differential equations, and the generative process involves inverting this diffusion process to generate samples. The training process involves minimizing the negative log-likelihood of the training data. The diffusion process involves iteratively adding Gaussian noise to the signal, and then applying a transformation to the signal to smooth out the noise. The generative process involves inverting this diffusion process to generate new samples that are similar to the training data.

Overall, GANs, VAEs, and Diffusion Models are all powerful generative models that can be used for a variety of applications. GANs are particularly effective at generating realistic-looking images, while VAEs are well-suited for generating structured data such as text and music. Diffusion Models have shown promise in generating high-quality images and other types of signals.

In summary, generative modeling is a subfield of machine learning that focuses on building models that can generate new data samples that are similar to the data that they were trained on. GANs, VAEs, and Diffusion Models are all popular generative models that each have their own strengths and weaknesses. By understanding the differences between these models, researchers can choose the right model for a particular application and achieve high-quality generative results.

1.4.1.2 Super-resolution

Super-resolution refers to the process of increasing the spatial resolution of an image beyond the limit imposed by the physical sensor or imaging system. We employ super-resolution for terrain enhancement in chapter 3 and chapter 5.

One of the most popular approaches for super-resolution is based on the use of convolutional neural networks (CNNs). In this approach, a CNN is trained to learn the mapping between low-resolution and high-resolution images. The CNN takes a low-resolution image as input and outputs a high-resolution image that is as close as possible to the ground-truth high-resolution image. The training is typically done using pairs of low-resolution and high-resolution images, with the aim of minimizing the difference between the output of the CNN and the ground-truth high-resolution image.

Mathematically, super-resolution can be formulated as follows. Let x be the low-resolution signal, y be the high-resolution signal, and H be the up-sampling operator that maps x to y. Then, the super-resolution problem can be written as:

$$x^* = \text{argmin}_x ||y - Hx||^2 \tag{1.3}$$

Various methods have been proposed for solving the super-resolution problem, including bicubic interpolation, iterative algorithms, and deep learning based approaches. The deep learning based approaches have shown state of the art performance in recent years, achieving high quality super resolution results on a wide range of images.

Overall, super-resolution is an important topic in image processing and computer vision, with applications in fields such as remote sensing, microscopy, and medical imaging.

1.4.2 Computer Graphics Landscape

Computer graphics techniques play a critical role in virtual world creation by enabling the generation and rendering of realistic and visually appealing environments. Some of the common computer graphics techniques used in virtual world creation include 3D modeling, texturing, lighting, and rendering. 3D modeling involves creating digital representations of objects such as trees, while texturing adds details and surface characteristics to these models. Lighting is used to simulate realistic lighting conditions in the virtual environment, while rendering produces the final image or animation by processing the 3D models with textures and lighting. These techniques are used in various applications, including video game development, virtual reality experiences, and architectural visualization, to create engaging and immersive virtual worlds.

1.4.2.1 Rendering

Rendering refers to the process of generating an image from a 3D model or scene by simulating the behavior of light and other physical phenomena. It is a crucial component of computer graphics, and is used in a wide range of applications including video games, film and television, architecture and design, and scientific visualization.

Rendering typically involves several stages, including modeling, texturing, lighting, and shading [101]. During the modeling stage, the 3D geometry of the objects in the scene is created or imported from a file. Texturing involves applying 2D images to the surfaces of the 3D models in order to give them color and detail.

Lighting [52] is one of the most important aspects of rendering, as it determines how the objects in the scene will be illuminated. Different types of lighting can be used, including directional lights, point lights, and ambient lighting. In addition to lighting, shading is used to determine how light interacts with the surfaces of the objects in the scene, taking into account factors such as reflection, refraction, and absorption.

Once all of the components of the scene have been created and configured, the rendering engine calculates how light interacts with each object in the scene, and generates an image that represents what the camera would see if it were positioned in the scene. This image can be further processed or post-processed in various ways to create the final output.

One of the most important factors in rendering is the choice of rendering algorithm. Different algorithms have different strengths and weaknesses, and the choice of algorithm will depend on factors such as the desired level of realism, the size and complexity of the scene, and the available hardware resources. For example, real-time rendering algorithms such as rasterization [104] are optimized for speed and interactivity, while offline rendering algorithms such as ray tracing [105] and path tracing [103] are optimized for accuracy and realism.

Overall, rendering is a complex process that requires a combination of artistic skill and technical knowledge. Advances in rendering technology have enabled increasingly realistic and immersive virtual environments, and have opened up new possibilities for creative expression and scientific visualization.

1.4.2.2 Shaders

Shaders [53] are an essential part of computer graphics, providing the means to define how light interacts with surfaces and creating the visual effects that bring digital images and animations to life. They are small programs written in specialized languages that run on the graphics processing unit (GPU) of a computer, and they can be used to create a wide range of visual effects, from realistic lighting and shadows to abstract shapes and patterns.

There are several types of shaders used in computer graphics, each with its own specific purpose. Vertex shaders are used to manipulate the geometry of 3D objects by transforming their vertices. Fragment shaders, also known as pixel shaders, control how the color and other properties of individual pixels are computed. Geometry shaders [51] operate on entire primitives, such as triangles or lines, and can be used to generate new primitives or modify existing ones. Tessellation shaders [54] are a relatively new type of shader that can be used to dynamically subdivide geometry into finer detail, allowing for more detailed and organic shapes. Tessellation shaders can be used to create highly detailed organic shapes, such as landscapes or human faces, that would be difficult or impossible to create using traditional polygonal modeling techniques. Compute shaders [50] are a more general-purpose type of shader that can perform complex calculations on data without necessarily generating any visible graphics.

Shaders are typically written in specialized languages such as OpenGL Shading Language (GLSL) [39]. These languages are designed to be highly optimized for use on GPUs and to allow developers to write code that can run in parallel on multiple cores.

In summary, shaders are a crucial component of modern computer graphics, providing the means to create a wide range of visual effects and simulations. They require specialized knowledge and tools to develop, but can be used to create stunning and realistic graphics that would be impossible to achieve with traditional rendering techniques.

1.4.2.3 Tools Used

OpenGL OpenGL is a cross-platform graphics API (Application Programming Interface) that provides developers with a set of functions to create interactive 2D and 3D graphics applications. OpenGL

is widely used in video games, scientific visualization, CAD, and virtual reality applications. It is also supported on a variety of platforms, including Windows, Linux, and macOS. Some of the key features of OpenGL include support for advanced lighting and shading techniques, hardware-accelerated rendering, and the ability to render large models in real-time.

Blender Blender is a 3D computer graphics software that is used for creating animated films, visual effects, 3D models, and video games. It is an open-source software that is available for free and is supported on Windows, Linux, and macOS. Blender features a powerful set of tools for modeling, sculpting, texturing, and animating 3D objects. It also has a built-in game engine that allows developers to create interactive 3D games. Other notable features of Blender include support for a variety of file formats, GPU rendering, and the ability to create simulations such as fluid and smoke simulations.

Terragen Terragen is a landscape generator and rendering software that is used for creating photorealistic landscapes and natural environments. Terragen is often used in film and television productions, as well as in video games and architectural visualization. Terragen can generate a wide range of natural environments, including mountains, deserts, forests, and oceans. It also features a variety of tools for terrain sculpting, vegetation placement, and atmospheric effects such as clouds and fog. Other notable features of Terragen include support for advanced lighting and shading techniques, GPU rendering, and the ability to create large-scale landscapes.

1.4.3 GIS Landscape

Geographic Information Systems (GIS) [100] is a powerful tool that can be used to create virtual worlds that simulate real-world landscapes and environments. Two key components of GIS that are essential for virtual world creation are Digital Elevation Models (DEMs) and satellite imagery.

A *Digital Elevation Model (DEM)* [99] is a representation of the topography of a landscape. It is a digital dataset that describes the elevation of points on the surface of the earth, typically at regular intervals (e.g., every 30 meters or every 1 arc-second). TINs, on the other hand, are made up of a set of non-overlapping triangles, each with its own set of vertices and elevation values. Other data structures like point clouds and level sets are also used for terrain representation. Terrain data can be captured through various methods, including LiDAR, photogrammetry, and ground surveying.

Airborne lidar [102] is a popular method for capturing DEM data because it provides high spatial resolution and accuracy. Lidar uses laser pulses to measure the distance between the sensor and the ground, creating a point cloud that can be used to generate a DEM. Ground-based surveying involves physically measuring elevation at various points on the ground, which can be time-consuming and expensive. Satellite-based radar is another method for capturing DEM data, but it typically has lower resolution and accuracy compared to lidar.

Satellite imagery is another important component of GIS for virtual world creation. Satellite imagery can be used to create realistic textures and colors for virtual landscapes. There are a variety of satellite

sensors that capture imagery at different spatial and spectral resolutions. For example, Landsat satellites capture imagery at 30-meter spatial resolution, while high-resolution commercial satellites can capture imagery at resolutions as fine as 30 centimeters.

The *US Geological Survey (USGS)* [96] is a key provider of DEM and satellite imagery data for GIS applications. The USGS provides a web-based portal for accessing and downloading a variety of GIS data, including DEMs and satellite imagery. The USGS provides free and open access to Landsat imagery. In addition, the USGS provides a variety of other GIS data, such as hydrography, transportation, and land cover.

Other related topics that may be relevant for virtual world creation using GIS include geospatial analysis, 3D modeling, and spatial data management. Geospatial analysis involves using GIS tools to analyze and manipulate geospatial data, such as creating slope or aspect maps from DEM data. 3D modeling involves using GIS data to create realistic 3D visualizations of virtual landscapes. Spatial data management [22] involves organizing and storing geospatial data in a way that facilitates efficient analysis and visualization.

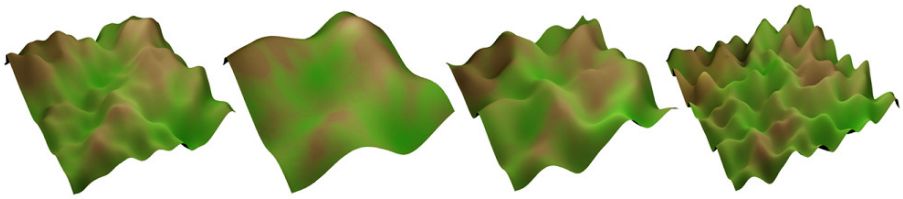
Overall, GIS is a critical tool for virtual world creation, and DEMs and satellite imagery are two key components of GIS that are essential for creating realistic virtual landscapes. The USGS is a key provider of GIS data for virtual world creation, including DEMs and satellite imagery. Other related topics, such as geospatial analysis, 3D modeling, and spatial data management, may also be important for creating high-quality virtual worlds using GIS.

1.5 Background and Related Work

1.5.1 Terrain Generation

Terrain generation can broadly be categorised into traditional and learning based approaches. We can further subdivide traditional approaches into noise-based, example-based and simulation-based techniques. On the other hand, we divide learning based methods based on the applications and techniques they use.

Noise based approaches are among the most commonly used methods due to their simplicity. Noise based functions are mainly of two types, value noise in which we interpolate between random values at fixed grid points and gradient noise in which we interpolate between random gradients in fixed grid points. Examples of noise based approaches are Perlin Noise [75, 76], Simplex Noise [24] or diamond square algorithm [15]. Perlin noise is a type of gradient noise and has been used extensively to generate terrains [72, 56] due to its efficiency and simplicity. Perlin noise uses random gradient values at grid points. For any point within the square grid, we calculate the dot product of the directional vector from the grid corner to the point with the random gradient vector on the grid. We then smoothly interpolate between the dot products to determine the elevation value at the given point. Let us denote Perlin noise by noise(x, y), such that we get an elevation value at sampled $x, y \in \mathbb{R}^2$. We use fractional Brownian



(a) (b) (c) (d)

Figure 1.9: Illustration of fractal Perlin Noise along with its octaves. For this rendition, $f_p = 2.0$, $a_p = 0.5$ and $N_o = 3$.

motion (fBm) [62, 74] to fuse multiple frequencies of Perlin noise. Let us denote the frequency of fBm for Perlin noise by $f_p \in [1, \infty)$, persistence $a_p \in [0, 1]$ and the number of octaves as $N_o \in [1, \infty)$. Then fractal Perlin noise is given by,

$$\sum_{i=1}^{N_o} a_p^i \times \text{noise}(f_p^i \times x, f_p^i \times y) \tag{1.4}$$

Figure 1.9 renders a terrain corresponding to Equation 1.4. Figure 1.9 (b), (c) and (d) correspond to the sum terms for i=1,2 and 3 called octaves o_1, o_2 and o_3 of Perlin noise respectively whereas Figure 1.9 (a) is the fBm summation of those terms as given by Equation 1.4. This equation relies on the fractal property of terrains, that is terrains exist as self-repeating structures at different scales and frequencies.

Simulation based methods consist of erosion tools [67, 8, 9], such that given a terrain, an erosion process is simulated over it to produce natural terrain patterns. Whereas example based methods use examples or sketches of terrains to generate terrains [117, 29].

Learning based methods for terrain generation were primarily dominated by GAN [20, 65, 35] and VAE [43] based methods. Conditional GANs [65] are an extension of GAN which learn a conditional distribution. Pip2Pix [35] is a conditional GAN image to image network which have been employed by [23] to convert user inputs such as ridges and valley into terrain. Similarly, [116] used an adversarial loss to amplify terrain with style components.

Image completion, inpainting, and outpainting have gained attention in recent years due to the advancements in deep learning techniques. Image completion [34] and inpainting [73, 108] involve 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. Outpainting [109] involves generating new content beyond the boundaries of the input image that can be used to generate high-resolution images from low-resolution inputs, as well as to create panoramic images from a single image.

1.5.2 Terrain Enhancement

Terrain enhancement involves adding details to the terrain. Traditionally, this has been achieved through the use of simulation erosion models [67, 106] on top of generated terrain, which can be time-

consuming despite adhering to physical laws. In recent years, deep learning methods have become popular for terrain enhancement, with techniques such as [23] seeking to learn erosion features to accelerate the process.

Terrain enhancement can be framed as a terrain super-resolution problem, wherein low-resolution terrain DEM is converted to a higher resolution. Methods such as [3, 46] employ ortho-photo and depth map pairs to super-resolve low-resolution terrain patches using techniques such as attention feedback networks. Other methods, such as [47, 36] use only the depth map to super-resolve terrain. Other enhancement techniques such as [116] enhance while incorporating style embeddings.

Progressive super-resolution [48, 49, 97] is a type of image super-resolution technique that involves creating a series of intermediate images with increasing resolution until the desired final resolution is achieved. This intermediate image is used as input to generate another image with a higher resolution, and the process is repeated until the final desired resolution is achieved. The advantage of this approach is that it can produce high-quality images with much higher resolution than the original input image.

1.5.3 Tree Generation

There exist many classes of methods to model trees. Some of the classes are parametric methods, image-based methods, modelling through 3D reconstruction, learning-based methods, grammar-based methods, etc.

Parametric methods are often much harder to understand and work with. Weber and Penn [98] introduced a system to model trees using a parametric approach. Their method is implemented in blender as a famous plugin *Sapling Tree Gen*. [98] also proposed additional use cases in their work such as pruning of trees, wind sway, vertical attraction and tropism in trees, etc.

Image-based modelling techniques [95, 26, 78] are used to model the most realistic trees but cannot fit many use cases because of lack of scalability. Some methods such as [89] lie at an intersection of image based approaches and grammar based approaches to model trees.

[95] proposed a model using an image-based approach. Their method used structure from motion and other post-processing techniques to model trees. Other methods exist such as single image approaches [26] and multiple image approaches [84], [78]. Methods of 3D reconstruction such as [110] use an exemplar database consisting of real trees reconstructed from scanned 3D data. This approach similarly lacks scalability due to the difficulty in the acquisition of data. Additionally, in use cases like modelling forests, these methods will not work because a single set of 20 to 30 images produces a single tree and these models lack stochasticity to produce variations in those trees. Producing a forest may require a big database of images and computational needs can be too expensive. More importantly, the stem structure information might be incomplete due to occlusion of leaves and hence it is not easy to animate such reconstructed trees.

Learning-based methods have been proposed to model trees. One such strategy is inverse procedural modelling [111], [80], [92], [91] in which the parameters of grammar-based models or parametric models are estimated given the data such as tree models. Other learning-based approaches such as [2]

take images as input but mainly focus on 2D trees, though they have a limited extension to 3D. Further work on Interactive procedural modelling like [60] provided for manual interaction for the 3D artist reducing their design process. Whereas [12], [38] optimise procedural modelling approaches at scale to efficiently generate forest for virtual world creation at reduced time and memory requirements. On the other hand, [40] presented a visually optimised method to make tree generation and rendering more suitable for applications in VR/AR.

Grammar-based approaches, such as the proposed work is based on the interpretation of the grammar provided by the user. Different methods of interpretation vary the expressibility and the number of species that can be modelled. L-system and its extensions [57] are popular grammar-based method to model trees. In [57], the grammar is first expanded and then interpreted to give results. Another grammar-based approach was proposed by [93]. Their model is easy for a user to understand thereby giving the user bigger flexibility to model trees but is not able to model a wide variety of trees. The limitation on the variety of trees is because all of their geometric parameters are set globally, i.e., it is the same for all types of stems. This makes it harder to model local variations present in trees. The proposed model overcomes this limitation by allowing the user to specify local variations in the tree structure.

1.5.4 Terrain Rendering

Terrain rendering refers to the process of creating a visual representation of a three-dimensional terrain. There exist many terrain rendering techniques in the literature [71, 31, 112, 59], with some popular ones being ROAM [14], Geometrical MipMapping [10], Geometry Clipmaps [61] and quadtree based.

The ROAM algorithm [14] is a technique used for real-time mesh adaptation that maintains continuous triangulations with the aid of priority queues. It utilizes pre-processed bintree triangles to enhance its performance. Geometrical MipMapping [10] is a block-based terrain rendering algorithm that reduces CPU processing time by dividing the terrain into grid-based tiles. The algorithm is designed to focus on GPU hardware-based rendering and minimize CPU overhead. Geometry Clipmap [61], is another efficient algorithm for caching terrain in nested regular grids around the viewer, providing visual continuity, uniform frame rate, and graceful degradation.

Quadtree-based terrain rendering [70, 113, 107, 83] is a another popular approach for rendering large-scale terrains. A quadtree is a hierarchical data structure that recursively divides a 2D space into four equal-sized quadrants. The basic idea behind quadtree-based terrain rendering is to divide a large terrain into a set of smaller tiles, each of which is represented by a node in a quadtree data structure. The terrain is initially represented by a single node at the highest level of the tree. This node is recursively subdivided into four child nodes, each of which represents a quadrant of the terrain. This process is repeated recursively for each child node until a desired level of detail is reached. Quadtree-based terrain rendering also allows for efficient culling of non-visible portions of the terrain, since each quad can be

individually culled. During rendering, the quadtree is traversed starting from the root node, and tiles that are visible in the current view frustum are recursively subdivided and rendered.

1.6 Thesis contribution

(a) Generated terrain

(b) Generated trees

Figure 1.10: Terrain and trees generated using our methods.

The primary focus of this thesis is to address the challenges of generating terrain and vegetation components of virtual worlds as shown in Figure 1.10. Specifically, this thesis will explore the use of learning-based generative approaches for terrain generation and L-systems for tree generation. The goal is to develop efficient and effective methods to generate visually realistic and varied terrain and vegetation. Furthermore, this thesis also aims to address the challenge of real-time rendering of virtual worlds. Achieving real-time rendering of complex virtual environments is computationally demanding. Therefore, this thesis aims to investigate novel rendering techniques which can be seamlessly integrated with generative algorithms and achieve real-time performance while maintaining high visual fidelity.

The expected outcomes of this thesis are to develop a system for generating and rendering virtual worlds that can be used in various applications, such as gaming, simulation, education, and training. The system should be efficient, scalable, and able to handle different types of terrain and trees. The system should also be flexible, allowing for customization and modification by users. By addressing these challenges, this thesis aims to contribute to the development of more efficient and effective techniques for creating virtual worlds. The chapter-wise enumeration of our contributions are as follows:

1. **Deep Generative Framework for Interactive 3D Terrain Authoring and Manipulation** [68]
    - Proposal of a novel terrain authoring framework that utilizes a combination of VAE and conditional GAN model to generate multiple variants of terrain from a single input.
    - Learns a latent space from a real-world terrain dataset.

- Presents an interactive tool that enables users to generate diverse terrains.

2. **Adaptive & Multi-Resolution Procedural Infinite Terrain Generation with Diffusion Models and Perlin Noise** [36]

    - A novel adaptive multi-resolution framework for generating terrains that combines a diffusion based generative network and frequency-separated terrain features.
    - SOTA terrain super-resolution for enhancing generated terrain patches.
    - Novel kernel-based blending method for generating infinite terrain with realistic terrain features using Perlin noise.

3. **Automated Tree Generation Using Grammar & Particle System** [37]

    - Grammar-based solution for 3D tree generation that can model a wide range of species improving the interpretability and expressibility of L-systems.
    - Adopts a particle system approach for foliage, models the tree growth variations, changes in foliage across seasons, and leaf structure.
    - Blender add-on developed for use, which can be used for efficient 3D tree generation along with a library of Indian and Western tree species.

4. **Real-Time Terrain Generation and Rendering**

    - A learning based framework for infinite terrain generation and level of detailing.
    - A rendering algorithm seamlessly integrated with the generative framework.
    - Application developed using OpenGL and GLSL with quad-tree based data management, added with botanical trees for aesthetics and user immersion.

1.7 Thesis organisation

This thesis is structured into several chapters that address different aspects of creating a virtual world. Chapter 1 established the problem and motivation for the research, as well as provided a background overview of the necessary topics. Chapter 2 focuses on generating and manipulating terrain patches, which form the basis of the virtual world. Chapter 3 extends this concept to create infinite terrains through an adaptive and multi-resolution approach. Chapter 4 covers the generation of trees, which enhances the realism of the virtual world by improving the interpretability and expressibility of L-systems. Chapter 5 builds upon the previous chapters to create a fully realized virtual world, complete with generated terrain and trees, rendered through a seamlessly integrated algorithm. The concluding chapter, Chapter 6, provides a summary of the research and discusses potential future directions for further study.

Chapter 2

Deep Generative Framework for Interactive 3D Terrain Authoring and Manipulation

2.1 Introduction

In the previous chapter we introduced the problem and goals. In this chapter, we continue our discussion by introducing the generation and manipulation of the fundamental building blocks of our virtual world - terrain patches. 3D Terrain modelling aims to create a digital representation of the real-world topography and is useful in both scientific applications surrounding land surface processes like flooding, soil erosion, as well as virtual terrain rendering in graphics and computer vision applications. It is also most sought for by the multimedia applications like Virtual Reality (VR) models and gaming. The real-world terrains undergo a range of natural transformations such as erosion, weathering, and landslides over the years, leading to the formation of complex topographies such as hills, mountain ranges, canyons, plateaus, and plains. This makes 3D terrain generation and authoring a challenging task. Existing 3D terrain authoring and modelling techniques have addressed some of these and can be broadly categorised as *procedural modelling, simulation method*, and *example-based methods* (refer to [18] for a detailed survey).

Recent advancements in deep learning have enabled us to learn diverse terrain features for tasks like terrain amplification [46], modifications [85], etc. In the context of deep learning-based automated 3D terrain authoring, the literature is very sparse. One of the most relevant example-based 3D terrain authoring methods referred to in this work as *TSynthNet* [25], trains a conditional Generative Adversarial Network (cGAN) on a large set of real-world terrain data to generate realistic virtual terrains from hand-drawn user inputs. However, they provide limited user control and generate a single terrain for a given input. Additionally, their method allows the use of either drawing of level-sets or ridge/valley strokes (with altitude cues). However, these two representations are complementary and jointly provide richer information for terrain authoring tasks.

In this chapter, we propose a novel realistic 3D terrain authoring framework powered by a combination of Variational Auto-encoder (VAE) [44], and conditional GAN model. Our framework attempts to

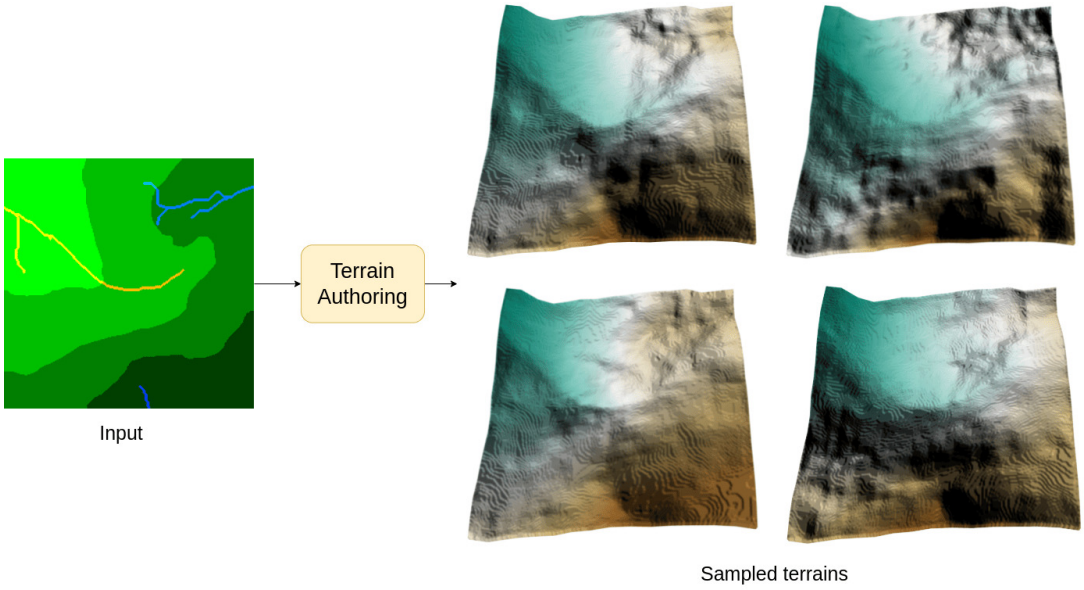


Figure 2.1: Generation of multiple variants of terrain from a single input topographic map sketch.

overcome the limitations of TSynthNet by learning a latent space from a real-world terrain dataset. This latent space allows us to generate multiple variants of terrain from a single input (as depicted in Figure 2.1) as well as interpolate between terrains while keeping the generated terrains close to real-world data distribution (Figure 2.2).

We also design a novel VAE loss function to exploit sparse topographic features like ridge/valley lines. Finally, we developed an interactive tool that lets the user model diverse 3D terrains with minimal inputs. We perform a thorough qualitative and quantitative analysis and provide a comparison with TSynthNet to show the superiority of our method over SOTA methods. Our codebase[1] and project page[2] are publicly accessible.

2.2 Methodology

We propose a two-stage framework to learn the topographical structure of real-world terrains from existing datasets and generate plausible DEM from input sketches that can be thought of as *topographic maps* primarily consisting of DEM level-sets and ridge/valley lines representing underlying abstract topological features of the terrain. Learning such a latent space enables two key use-cases of the proposed method, namely, automated generation of multiple variants of terrain from a single user input sketch and interpolating between two terrains while keeping the generated virtual terrains closer to a

[1] `https://github.com/Shanthika/TerrainAuthoring-Pytorch`
[2] `https://shanthika.github.io/terrain_authoring/`

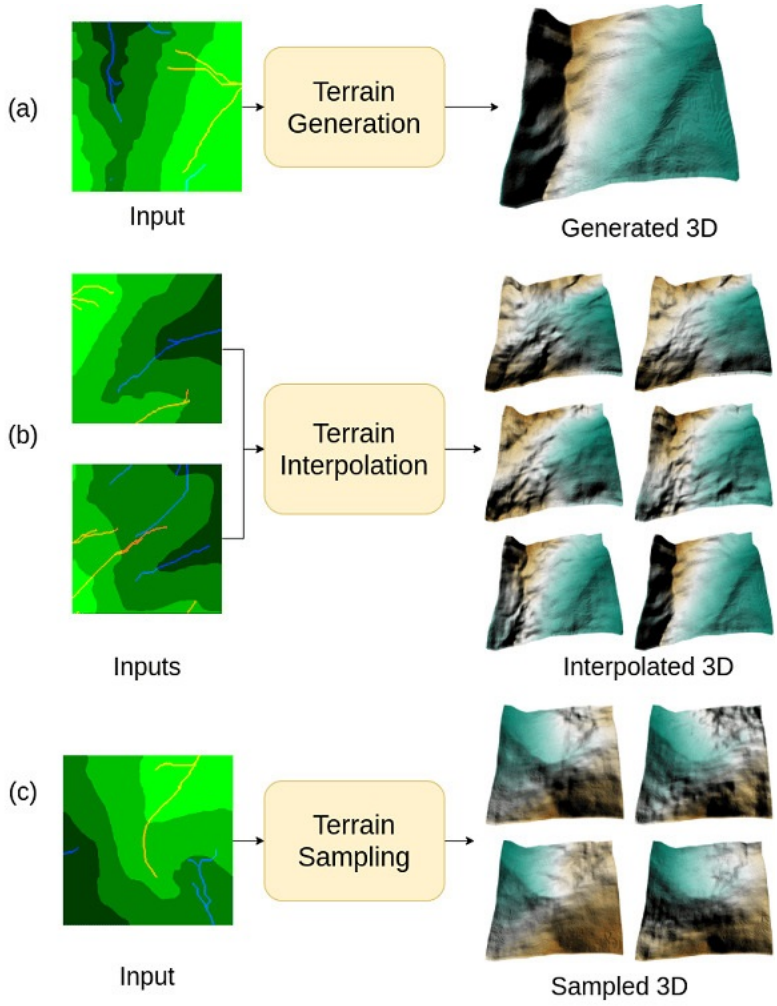


Figure 2.2: Terrain generation, interpolation and sampling by the proposed framework.

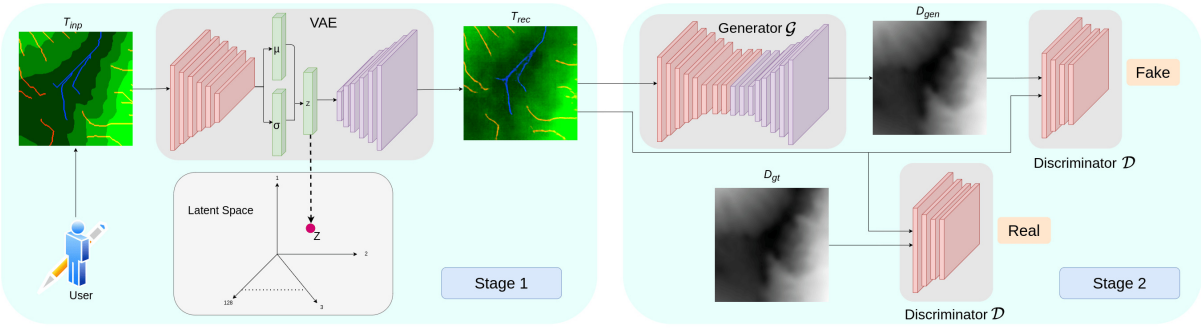


Figure 2.3: The architecture of the proposed two-stage deep generative framework.

real-world dataset consisting of realistic topographical features. Figure 2.3 provides an overview of the proposed two-stage framework.

2.2.1 Stage 1: Latent Space Learning

In the first stage, we aim to learn a generative latent space for topographic maps using a Variational Auto-encoder (VAE) model from a real-world terrain dataset. We extract ground truth topographic maps from real-world terrain data (see Section 2.3.1) and autoregress using VAE to learn the latent space. Let T_{inp} be our (hand-drawn) input sketches representing a rough topographic map. VAE learns to approximate a distribution $q(z)$ and learns the parameters μ and σ, from which the latent vector z is sampled using the re-parameterization trick as $z = \mu + \sigma * \epsilon$, where ϵ is sampled from a Standard Normal distribution. This sampled vector is fed to the decoder, which predicts T_{rec}, that is the reconstruction of original input T_{inp}.

We propose a novel auto-regressive reconstruction loss L_{recons} between VAE input T_{inp} and output T_{rec} by modifying the traditional Binary Cross Entropy (BCE) loss to emphasize the ridge/valley lines in the topographic map.

We propose to give higher weightage to the loss on red and blue channels to give more importance to ridge and valley lines/strokes in the topographic map sketches. Additionally, a traditional KL divergence loss L_{KL} ensure that the probability distribution of latent vector z follows a Standard Normal distribution. Thus, the final VAE loss L_{VAE} is a combination of reconstruction L_{recons} and KL divergence loss L_{KL} .

$$L_{VAE} = L_{recons} + \gamma * L_{KL} \tag{2.1}$$

The γ parameter in Eq.2.1 is the weighting of the latent loss L_{KL} which is set to 0.65.

2.2.2 Stage 2: DEM Generation

The second stage consists of a conditional Generative Adversarial Network (cGAN) (Pix2pix [35]) model that generates plausible DEM output. This stage aims to generate the DEM given user topographic map sketch or, in our case, generated sketch from the previous stage.

The overall network is trained such that both generator G and discriminator D reach a Nash equilibrium by playing a two-player minimax game while optimizing the value functions $V(G, D)$[35]. We use L1 loss for reconstruction from the generator, i.e., $L1(\mathcal{G})$. So the final loss for cGAN training is given in Eq. 2.2.

$$L = \min_{\mathcal{G}} \max_{\mathcal{D}} \left[V(\mathcal{D}, \mathcal{G}) + L1(\mathcal{G}) \right] \tag{2.2}$$

Table 2.1: Comparison with baseline and TSynthNet [25].

Method	RMSE ↓	PSNR ↑
TSynthNet [25]	5.391	31.875
Baseline (VAE)	9.229	7.322
Our model (VAE+cGAN)	**4.743**	**34.189**

2.3 Experiment Details

2.3.1 Dataset

We use a popular DEM dataset used by other relevant works in the literature, e.g., [3, 47] which is part of DEMs of mountain ranges named Pyrenees [33] and Tyrol [86], respectively. DEM patches with a resolution of 2m/pixel have been used as ground truth elevation maps. Original DEM tiles were split into 200x200 pixels. We randomly sample 3000 image patches for training and 878 image patches for testing. More details about the dataset can be referred from [47]. We prepare the training dataset by extracting the topographic map input sketches as RGB images from DEMs. Here the Green channel is dedicated to representing the elevation in the form of 4 level-sets while the Red(/Orange) and Blue channels are used to represent ridge and valley lines, respectively.

2.3.2 Implementation Details

Our VAE model is a 12 layer network with 6 layers in the encoder and 6 layers in the decoder. The latent space dimension is set to 128. All the layers consist of a 3x3 convolution with a stride of 2 and padding of 1, followed by Batch Normalization and using Leaky ReLU non-linearity. Adam optimizer was used to update the parameters with a learning rate of 0.001 and an exponential scheduler with gamma set to 0.95 while training the VAE.

Our conditional GAN generator is a U-net inspired Pix2pix architecture [35]. This model was trained using Adam optimizer with a learning rate of 0.0002, β_1 set to 0.5 and β_2 as 0.999 for both Generator and Discriminator. All our experiments were performed on a single Nvidia GTX 1080Ti.

2.3.3 Results

We provide quantitative evaluation results in Table 2.1. We can observe that we obtained superior performance with RMSE of 4.743 and PSNR of 34.189 from our model (VAE+cGAN) and beat the SOTA TSynthNet. We also demonstrate the results with the Baseline model (i.e., single-stage VAE based DEM generation) also performs inferior to our model, justifying need for a two-stage framework. Figure 2.4 shows the qualitative results where the first column shows the input and the VAE

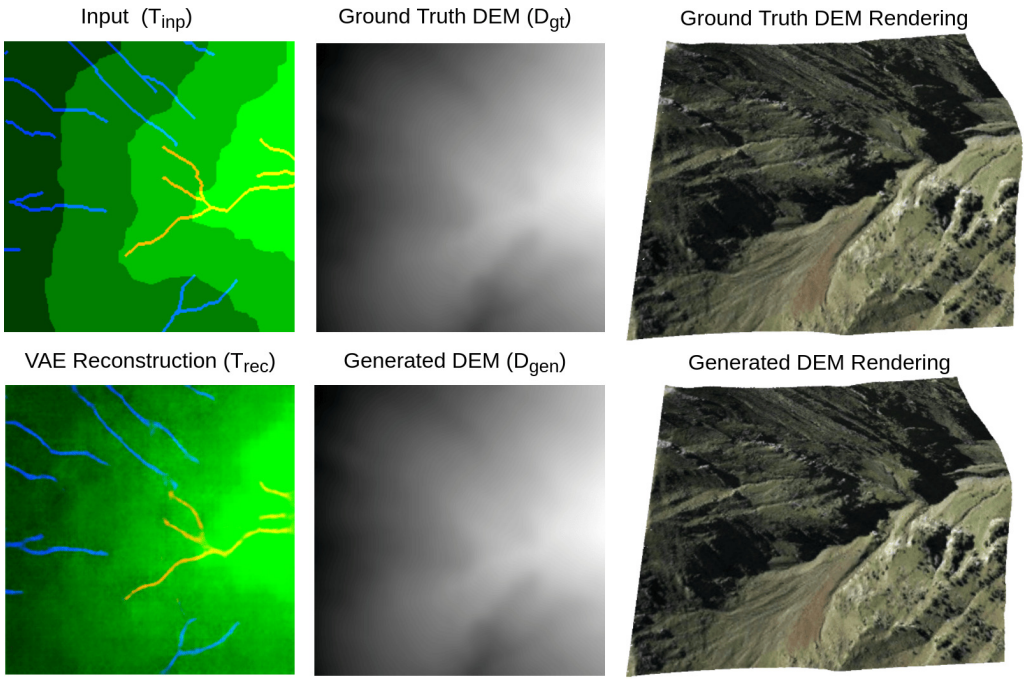


Figure 2.4: Rendering of generated and ground truth terrains for qualitative comparison.

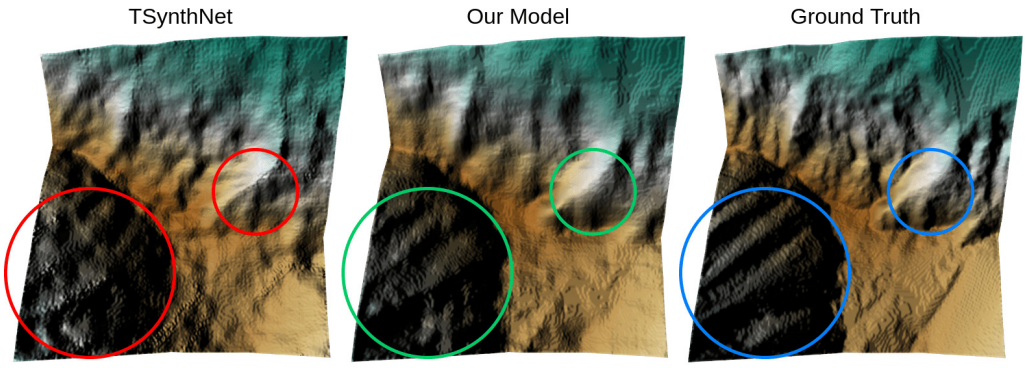


Figure 2.5: Qualitative comparison with TSynthNet [25].

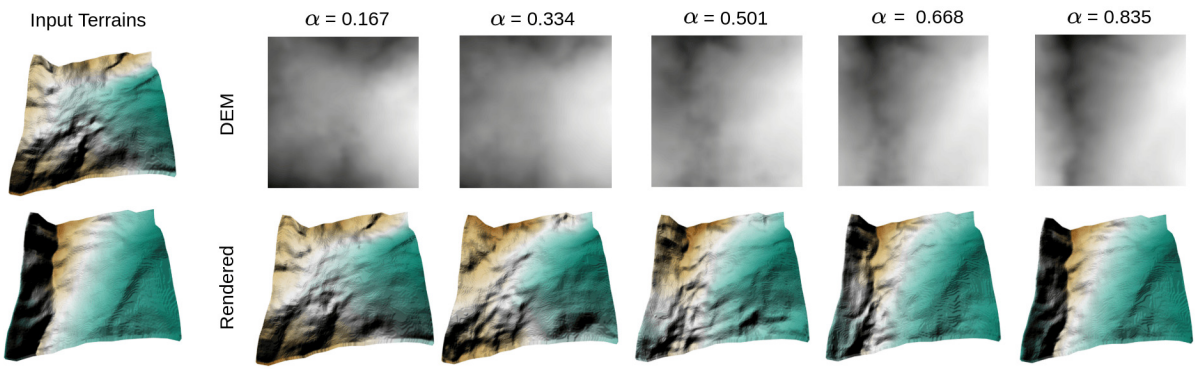


Figure 2.6: Terrain interpolation results for varying α parameters.

reconstructed topographic maps. The second column gives a comparison between the ground truth and generated DEMs. The last column shows the 3D rendering of these DEMs overlaid with the associated satellite image. Moreover, Figure 2.10 gives illustrates samples from our framework along with their 3D rendering.

We also provide a qualitative comparison of our method with TSynthNet in Figure 2.5. The red circles depict the region where TSynthNet deviates from ground truth terrain while our method (green circles) stick closer to the ground truth. Additionally, our method also enables terrain interpolation and variant generation using the learnt VAE latent space.

Generating Terrain Variants: We utilise the latent space created by VAE to generate different samples from the same input. Different terrains generated from the latent space encoding of the same input topographic map are shown in Figure 2.1. We can observe the generated terrains have realistic but slightly different topographical features from that of input terrain. This provides the user with the flexibility to generate multiple terrain DEMs and use them for large scale generation of virtual terrain maps. Furthermore, we can observe the variations in terrain structure for the same input in Figure 2.7 and for different input in Figure 2.8.

Terrain Interpolation: The latent space can also be used for automated fusion of topographic features across two terrains. Given two input DEMs we extract associated topographic map sketches and generate a new terrain by linear interpolation of the respective feature embedding (z_1 and z_2) in the VAE latent space. More specifically, we combine them using the formula $z = \alpha * z_1 + (1 - \alpha) * z_2$ while generating respective novel DEM using our framework. Figure 2.6 shows an example interpolation of two input terrains in the latent space for different values of α parameter.

2.3.4 User Study

We performed a user study involving 6 users. We presented users with a set of generated and ground truth terrain pairs overlaid with satellite images. The users were unable to decisively differentiate the

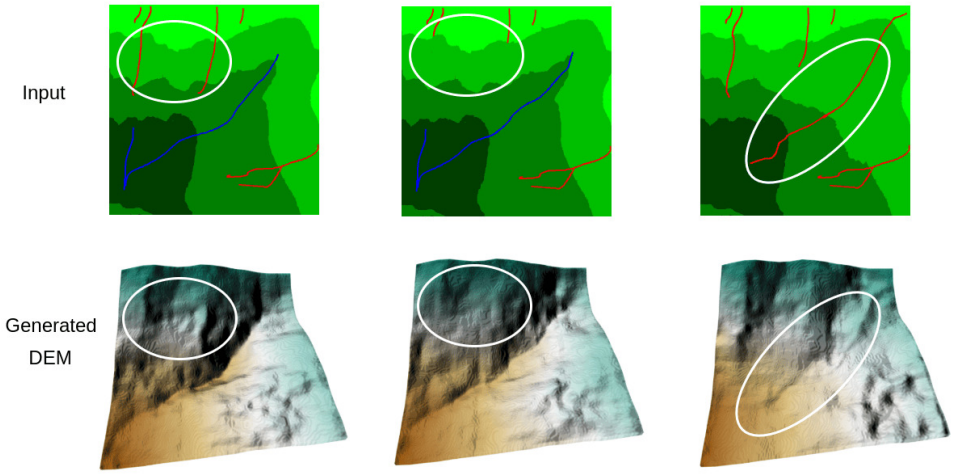


Figure 2.7: Variations in terrain generated for different input by the proposed framework.

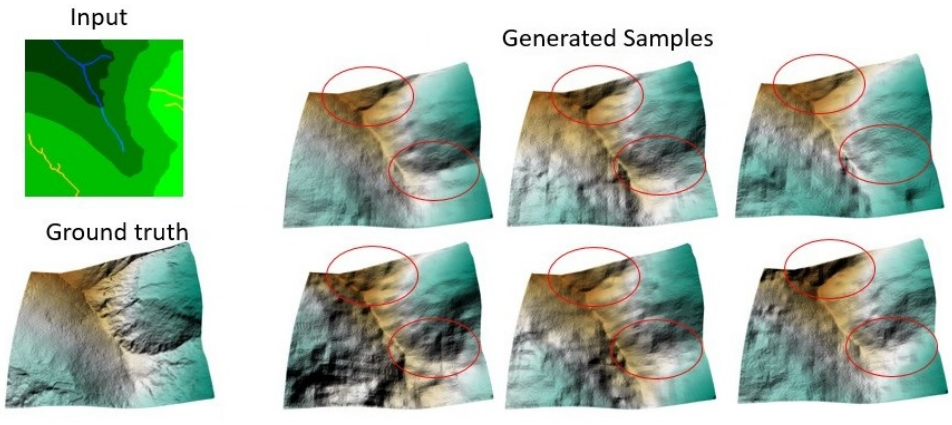


Figure 2.8: Variations in terrain generated for same input by the proposed framework.

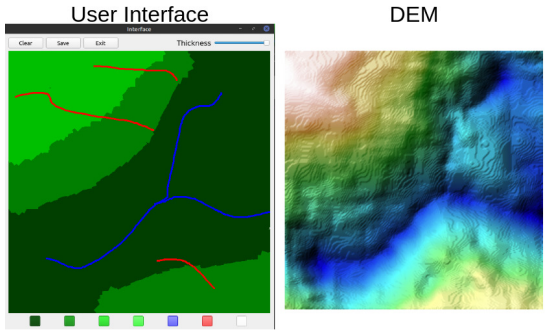


Figure 2.9: The UI developed for user study.

Table 2.2: User study experiment 2 results where the cells contain the percentage of people who responded with a score to our questions.

Questions	**Score 5**	**Score 4**	**Score 3**	**Score 2**	**Score 1**
Is the interface and application intuitive?	33.3%	66.7%	0%	0%	0%
Does the generated terrain follow the input?	50%	50%	0%	0%	0%
Was the UI fast and reactive?	100%	0%	0%	0%	0%
Is it easy to express ones intent?	50%	0%	16.7%	33.3%	0%

generated and ground truth terrains and choose the real terrain only 50% of the time. Total 83.3% of the users agreed that the terrains generated are very realistic, while 16.7% said that it is fairly realistic.

In the second experiment, we provide the user with a simple interface to draw input sketches. We provide the option to vary brush thickness so that the dense level-sets can be drawn with only a few strokes. The user interface and the DEM generated for a hand draw user input is shown in Figure 2.9. The input can also be interactively edited to get desired output. We asked the users several questions regarding the interface and the application. The results of the study are shown in Table 2.2 where the users had to rate from 1(lowest) to 5(highest). We observed that the users were able to generate DEMs with ease after a couple of attempts.

2.4 Conclusion

We proposed a novel realistic terrain authoring framework powered by a combination of VAE and conditional GAN model. Our framework learns a generative latent space from real world terrain dataset. This latent space allows us to generate multiple variants of terrain from a single input as well as interpolate between terrains, while keeping the generated terrains close to real world data distribution. While a preliminary interactive tool has been developed and used here, we further intend to provide user control to generate the terrain variants and interpolated terrains. The thorough qualitative and quantitative analysis and comparison with other SOTA methods support the superior outcome of our approach. In the next chapter, we continue the discussion of terrain generation by extending the idea of patches to generate infinite terrain.

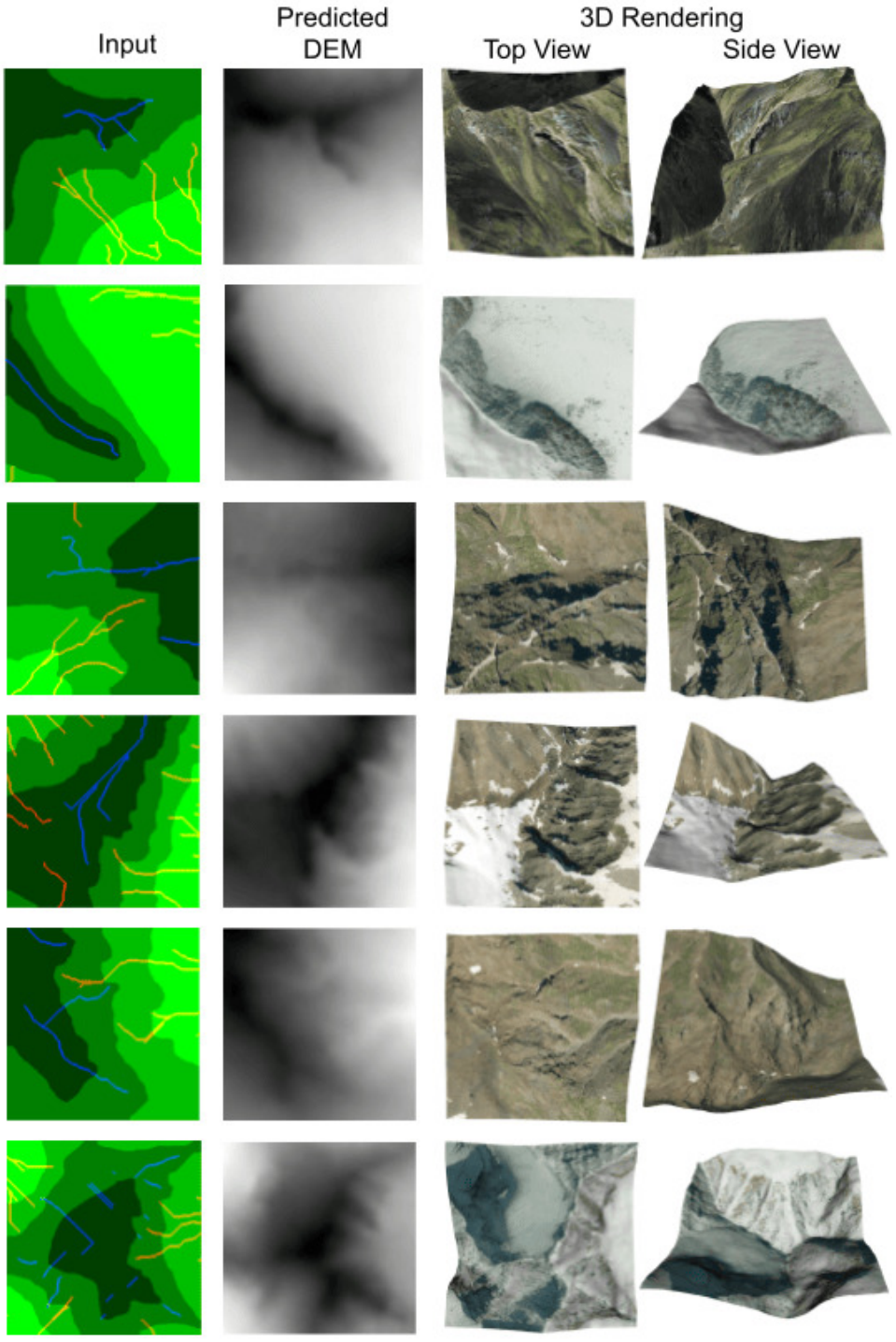


Figure 2.10: Terrain generated by the proposed framework along with 3D rendering of top and side view.

Chapter 3

Adaptive & Multi-Resolution Procedural Infinite Terrain Generation with Diffusion Models and Perlin Noise

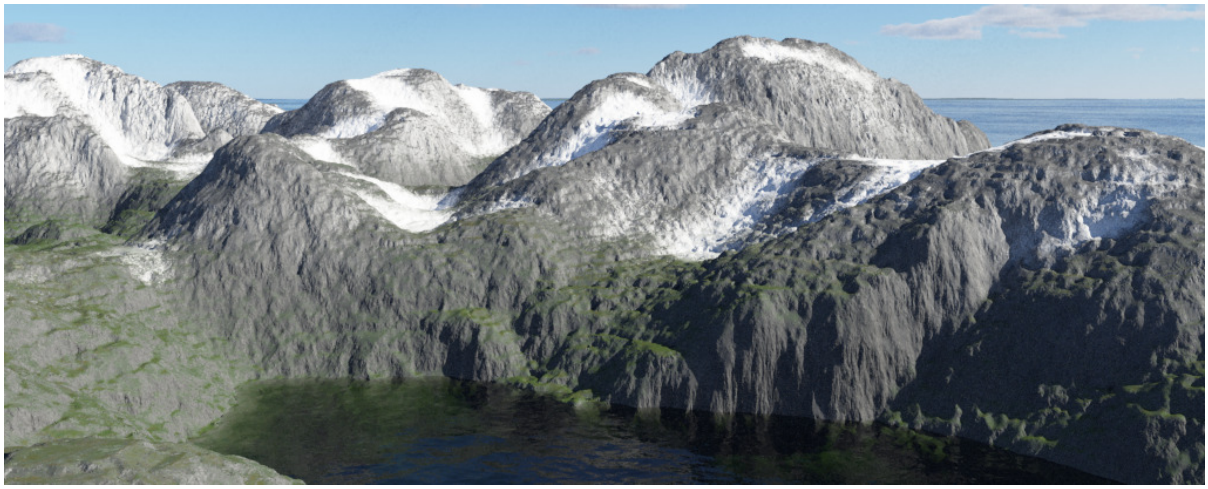


Figure 3.1: Rendition of a terrain generated using the proposed method.

3.1 Introduction

We continue our discussion in this chapter by extending the concepts of the previous chapter to generate infinite terrain. 3D Terrain generation is a classical use-case in the computer graphics community (dates back to four decades [16]) largely driven by gaming and simulation applications. Procedural and artist-driven 3D terrain generation are two popular lines of thought explored in the existing literature [82]. In procedural 3D terrain generation, we aim to algorithmically generate height maps of landmasses such as mountains or deserts. Many domains such as digital gaming, animated movies and architectural models adopt this approach, e.g., popular ”open-world” games such as *Minecraft* and *No Man's Sky* need to render infinitely large 3D terrains and hence employ procedural generation techniques. Further-

more, due to the large variation in hardware capability at the gamer's end, it is desired to have a solution with the ability to adaptively generated terrains of varying quality (i.e., multi-resolution terrains).

Traditionally, 3D terrain generation has been attempted using functions like Perlin [75] or Simplex [24] noise. This is to date a popular technique in digital gaming. However, these techniques provide minimal control largely restricted to choosing noise parameters and hence lack real-world terrain features. The real-world terrains are captured traditionally using microwave imaging (typically sensor over satellites) or recently using LiDAR-based sensing [66]. These digitized terrains are thus represented and stored as Digitial Elevation Models (DEMs) where a raster grid stores per pixel elevation of the discretized terrain.

With the advent of deep learning technology, many generative modelling approaches such as Generative Adversarial Networks (GAN) [20] and Variational Auto Encoders (VAE) [43] have been employed for the task of 3D terrain generation [23, 116] by learning over the real world DEM data. Their key advantage over traditional noise modelling techniques is that they enable learning from a large set of publicly available realistic terrain data [32, 19]. This brings realism to generated terrains. Nevertheless, their usage has been largely limited to producing limited size *tiles/patch* of terrain at a single resolution [68], which greatly limits their applications to open-world games and simulations where large-scale continuous infinite terrain generation is desired. Another related use-case of terrain enhancement/super-resolution is also well attempted with deep learning framework [47]. In regard to deep generative models, recently diffusion-based technique is getting popular where an iterative Markov modelling is proposed for learning data distribution and employed for the task of generating realistic images. The diffusion-based formalism is claimed to outperform other existing generative techniques [13].

In this chapter, we propose a framework to generate infinite 3D terrains at multiple resolutions adaptively. Our framework combines diffusion-based generative network [69, 13] and novel frequency separated 3D terrain features along with learnable terrain super-resolution equipped enhancement followed by novel Kernel-based Blending that uses Perlin noise [75] to generate infinite terrain with realistic terrain features. More specifically, our framework consists of training and inference phases. As part of the training phase, we propose to separate multiple spatial frequencies of 3D terrain features extracted from real-world data and independently employ diffusion model based learning of respective data distributions (at associated spatial frequency). Additionally, we also learn a 3D terrain super-resolution model over the same dataset in this phase. In the inference phase, we propose to adaptively sample learnt data distribution from diffusion models (at respective spatial frequencies) and perform fusion to obtain a new terrain patch. Subsequently, we enhance this patch using the trained super-resolution model to enhance realistic 3d terrain features into the patch. Finally, we combine multiple patches using novel kernel blending where Perlin noise help achieve seamless blending near patch boundaries enabling the generation of large and continuous realistic 3D terrains. Figure 3.1 shows a realistic 3D terrain generated with the proposed framework. We provide a comprehensive quantitative and qualitative evaluation.

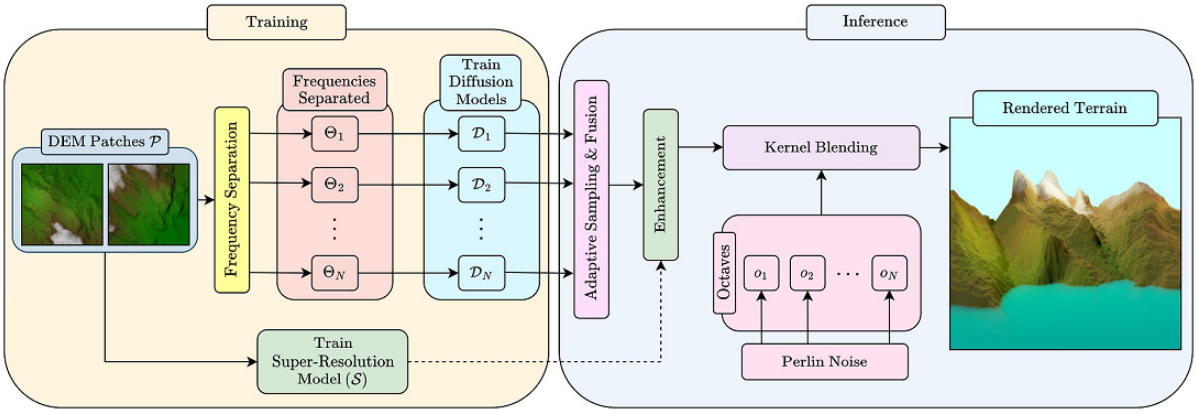


Figure 3.2: Overview of the proposed framework consisting of the training and inference phases.

Additional details and video resources can be found on our project page [1]. The code and model weights are publicly accessible [2,3].

The key contributions of our work are: 1) A novel adaptive and multi-resolution framework for generating realistic 3d terrains. 2) Diffusion-based generative modelling of 3d terrains. 3) Generating infinitely large 3d terrain in a learning-based framework. 4) State-of-the-art 3d terrain super-resolution technique.

3.2 Method

3.2.1 Overview & Notations

Our dataset consists of terrain represented as a Digital Elevation Model (DEM). The DEM is usually large and hence divided into a set of smaller patches for easy processing.

Figure 3.2 gives an overview of the proposed framework divided into two phases. In the training phase, we assume the availability of DEM dataset where we divide large DEMs into multiple smaller uniform size patches ρ_i and the set of all patches obtained from the dataset is represented as $\mathcal{P}$. Subsequently, we separate each patch $\rho_i \in \mathcal{P}$ into N of its constituent spatial frequencies $\mu_{1_i} \dots \mu_{N_i}$ using Fourier transform such that $\mu_{j_i} \in \Theta_j \forall i$. Hence, Θ_j is the set containing all patches of frequency j and N is a hyper-parameter of the framework. Furthermore, we resize the patches in $\Theta_1 \dots \Theta_N$ such that the size of the patches increases from 1 to N. Finally, we train N separate diffusion models $\mathcal{D}_1 \dots \mathcal{D}_N$ with $\Theta_1 \dots \Theta_N$, respectively. We also train a separate super-resolution model $\mathcal{S}$ on $\mathcal{P}$ to enhance generated patches.

[1] https://3dcomputervision.github.io/publications/inf_terrain_generation.html
[2] https://github.com/aryamaanjain/trcan
[3] https://github.com/aryamaanjain/perlin_noise

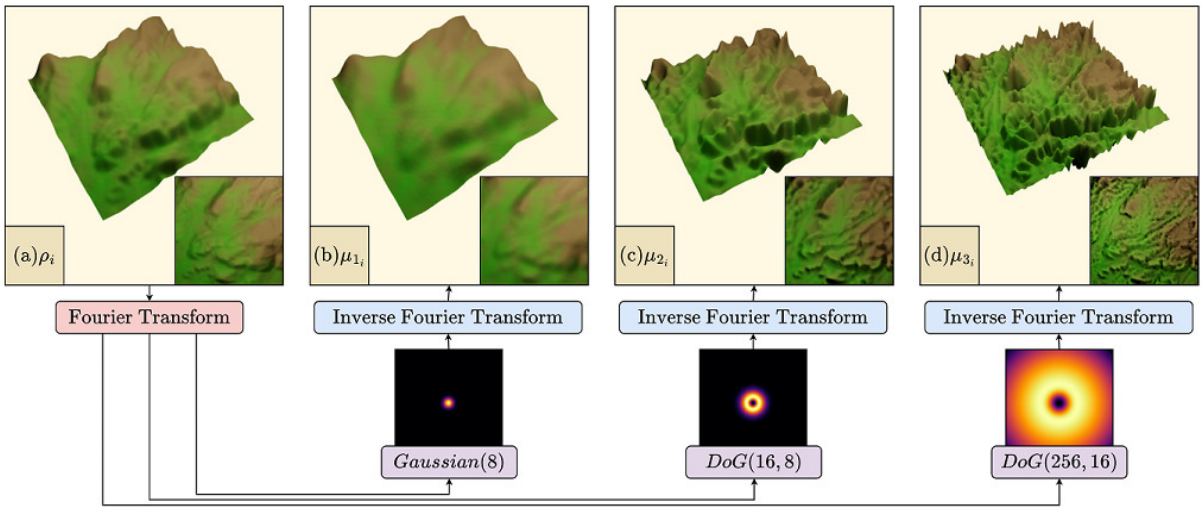


Figure 3.3: Separation of a patch into its constituent frequencies. The patch dimension is 256×256 and $N = 3$. The Gaussian kernel is of variance 8 and the Gaussians in the DoG kernel are of the variance are given in their parameters.

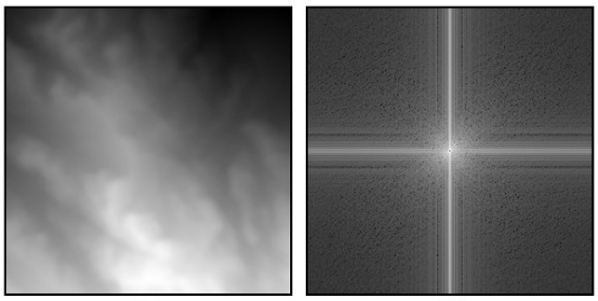
As part of the inference phase, we adaptively sample patches from the first k diffusion models, where $0 \leq k \leq N$. We can increase k adaptively upon the need for more details in the terrain, thereby providing a multi-resolution adaptive output. The sampled set of patches $\mu'_{1_i} \dots \mu'_{k_i}$ are further employed with bicubic up-sampling to be brought to the same size and fused together with fractional Brownian motion (fBm) [62] to generate patch ρ'_i. The generated patch is further enhanced by employing the trained super-resolution model ($\mathcal{S}$) to yield patch $\rho'_{i_{SR}}$. Finally, for a smooth transition between the generated patches $\rho'_{1_{SR}}, \rho'_{2_{SR}}, \dots$, we combine fractal Perlin noise with the generated patches using a derived kernel G. The detailed description of relevant modules in our framework is presented below.

3.2.2 Frequency Separation

We separate each patch $\rho_i \in \mathcal{P}$ into its N constituent frequencies $\mu_{1_i} \dots \mu_{N_i}$ using Discrete Fourier Transform (DFT) as show in Figure 3.3. DFT is an image processing method used to convert an image from spatial to frequency domain. We specifically use the Fast Fourier Transform (FFT) algorithm [7] $\mathcal{F}$ for our tasks. Given an input patch $\rho_i \in \mathcal{P}$ of size $M \times M$ in the spatial domain, we convert it into the frequency domain F of the same size as $F = \mathcal{F}(\rho_i)$ given by,

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} \rho_i(x, y) e^{-i\, 2\pi \left(\frac{ux}{M} + \frac{vy}{M}\right)} \tag{3.1}$$

Figure 3.4 illustrates the patch ρ_i and the corresponding log amplitude of its DFT. After we obtain the frequency domain representation F of the patch ρ_i, we apply a set of kernels to separate its frequencies at varying levels. We use a Gaussian kernel for the lowest frequency and Difference of Gaussian (DoG)



(a) (b)

Figure 3.4: (a) Patch (b) Corresponding log amplitude of the DFT of the patch.

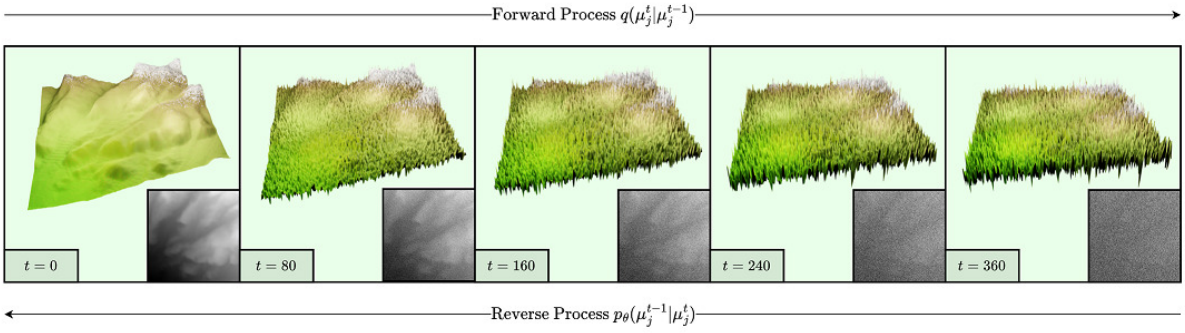


Figure 3.5: The working of a diffusion model for terrains. t represents the timestep of the diffusion process. Here we adopt linear schedule with β varying between 0.001 and 0.02.

kernels for higher frequencies as illustrated in Figure 3.3. These kernels mask F as Gaussian $\circ$ F or DoG $\circ$ F where $\circ$ represents the Hadamard product which aids in separation of the frequencies. After applying the kernels, we convert the resultant patches back to the spatial domain using Inverse FFT $\mathcal{F}^{-1}$ giving us the constituent patches at successively increasing spatial frequencies $\mu_{1_i} \dots \mu_{N_i}$. Separating a patch into its constituent spatial frequencies will aid in providing a multi-resolution and adaptive framework for producing terrains. One can observe that the extracted spatial frequency patches (shown in Figure 3.3) are conceptually very similar to the octaves depicted in Figure 1.9.

3.2.3 Diffusion Models

Diffusion models generate samples from a distribution by learning successive denoising starting from from a noisy sample at timestep T. Particularly, each sample in the order $\mu_j^T, \mu_j^{T-1}, \dots, \mu_j^1, \mu_j^0$ is closer to the desired distribution, where the superscript gives the timestep of the diffusion process. We adapt our model from [13] which consists of a forward process $q(\mu_j^t|\mu_j^{t-1})$ which adds noise to the given sample and a reverse process $p_\theta(\mu_j^{t-1}|\mu_j^t)$ which learn to denoise the sample. The reverse process is

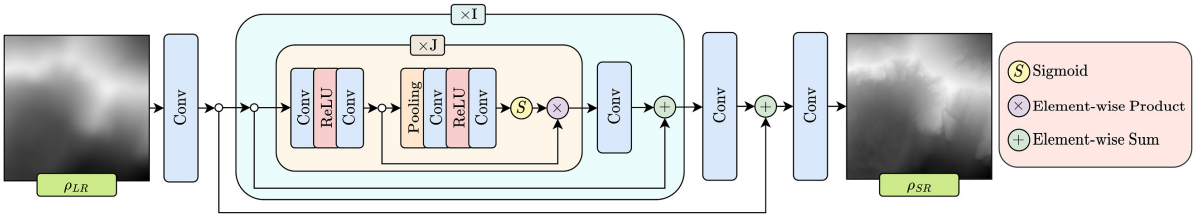


Figure 3.6: Architectural diagram of the proposed terrain super-resolution model TRCAN.

parameterized by θ which is learnt. For a given terrain frequency distribution sample $\mu_j^0 \sim \Theta_j$, we define the forward process as,

$$q(\mu_j^k|\mu_j^0) = \prod_{t=1}^{k} q(\mu_j^t|\mu_j^{t-1}) = \prod_{t=1}^{k} \mathcal{N}(\sqrt{1-\beta_t}\mu_j^{t-1}, \beta_t I)) \tag{3.2}$$

Here β is increased as per a linear or cosine schedule in our experiments. Since we iteratively scale and sample from a Normal distribution as illustrated in Equation 3.2, we get an isotropic Gaussian sample at timestep T. We define the reverse process as:,

$$p_\theta(\mu_j^{t-1}|\mu_j^t) = \mathcal{N}(\mathrm{M}_\theta(\mu_j^t, t), \Sigma_\theta(\mu_j^t, t)). \tag{3.3}$$

At each time step, we predict the mean and covariance of the previous timestep using the parameterized functions M_θ and Σ_θ parameterized by θ. We sample from a Normal distribution with the predicted mean and covariance iteratively until timestep 0, which gives us a sample from the desired terrain frequency distribution. We specifically predict the noise added, optimizing for the L_2 norm of the true and predicted noise $\mathbb{E}[||\epsilon - \epsilon_\theta(\mu_j^t, t)||_2^2]$, where ϵ is the true added noise and ϵ_θ is a parameterized function predicting the noise added. We use a UNet architecture.

We train N diffusion models $\mathcal{D}_i$, $i \in [1, N]$ with Θ_i. Each successive model is trained on a larger patch size, that is, the size of the patch produced by the model $\mathcal{D}_i \propto i$. In particular, we keep $N = 3$ in our experiments with the patch sizes varying as 64×64, 128×128 and 256×256 for $\mathcal{D}_1, \mathcal{D}_2$ and $\mathcal{D}_3$ respectively. Decreasing the patch size with decreasing frequency aids in adaptive sampling discussed in the sections to follow. We use diffusion models for learning the distribution of terrains because of their superior quality in terrain generation compared to other methods, as discussed in the section 3.3. An illustration of the process of terrain patch generation with diffusion model is provided in Figure 3.5.

3.2.4 Terrain Super-Resolution

Modelling: We propose to adapt RCAN [114] model initially proposed for image super-resolution for the task of terrain enhancement. Henceforth, we will refer to this network as Terrain Residual Channel Attention Network (TRCAN). Figure 3.6 provides an overview of the architecture of TRCAN. This network uses a residual in residual structure which helps in making very deep networks avoiding

vanishing gradients and increasing the receptive field. The input to TRCAN model while training is a low-resolution patch ρ_{LR} and the output is the super-resolved patch ρ_{SR} corresponding to the ground truth high-resolution patch ρ_{HR}.

The key difference between our method compared with RCAN [114] is the head and tail convolutional blocks in the network. Whereas RCAN proposes to pass the image in its low-resolution dimension through the network and upsample towards the end, we cannot employ the same for the task of terrain super-resolution. This difference primarily arises due to the difference in the ways of the processing of the datasets where terrains need the input and output to be of the same dimensions to find low-resolution and high-resolution patch correspondences. We subsequently modify the head and the tail convolutional blocks to account for the same.

We use Adam optimizer [42] along with L_1 loss to optimize the parameters of the network given by,

$$L_1(\theta) = ||\rho_{HR} - \rho_{SR}||_1^1 \tag{3.4}$$

Post-Processing: As compared to regular RGB images, DEMs are often large in resolutions, for example, Cimavertana region in our dataset is a 5250×3900 raster. Thus, the large DEMs are split into smaller patches for easy processing with typical sizes lying in the range 256×256. In the simplest form, all the patches are split without any overlap, enhanced independently and put together side by side without overlap to obtain the final super-resolved DEM. [47] observed that this was not the optimal way to obtain the final DEM and proposed to split the patches with some overlap along the edges (25% in their case), super-resolve and then average them along the edges. We observed that the scope of improving the results is not limited to just the patch boundaries but throughout the patch and therefore propose a new stride-based post-processing technique. We extract patches from the larger DEM at strides of s, which is a hyperparameter. We super-resolve all the patches extracted at stride s and pool their values at the overlapping regions to obtain the final result. This acts as an implicit ensemble thereby improving the RMSE of the super-resolved DEM. This can be seen as a generalization of the method proposed by [47], with a 25% overlap corresponding to a stride of 192 for a 256×256 tile. We call this model TRCAN+.

3.2.5 Adaptive Sampling, Fusion & Enhancement

We aim to produce a multi-resolution and adaptive framework to generate terrain. A naïve approach for generating terrains would be to learn the DEM patch $\mathcal{P}$ distribution rather than the distributions of its constituent frequencies Θ_j. In this approach, we would need to learn the patches at a constant resolution, and generate those patches at the same resolution regardless of any constraints. This might be wasteful of resources because high levels of detail for far away terrains may be unnecessary in a real-time setting. For example, in a first person view, we would like terrains close by to be of higher quality than areas of terrains far away. We would also like the details on the terrain to increase as we move closer begging a multi-resolution solution. Similarly, we would want an algorithm that could function on computational

constraints, that is work on PCs with lower specifications too. Therefore, we desire a solution which is adaptive, multi-resolution and realistic.

Firstly, we achieve adaptive sampling by varying the number of models we sample for generating a patch. Particularly, we sample $k \leq N$ diffusion models to produce $\mu'_1 \dots \mu'_k$. For example, in a first person view, we would set k high for a nearby point and keep it low for a far away point interpreting it as a technique for Continuous Levels of Detail (CLOD) in terrains. Another use case would be to set an upper-limit for k to respect computational constraints, with k set to 0 for the slowest PCs corresponding to just Perlin Noise which has efficient implementations available. Since for lower frequencies, the models are learnt on lower resolutions, sampling them would be faster. Once we obtain $\mu'_1 \dots \mu'_k$, we perform bi-cubic interpolation to bring all of the generated frequency separated patches to the same resolution and then fuse them using fBm, a method which is inspired by Perlin noise. Let the persistence of generated patches be denoted by $a_\mu \in [0, 1]$ which scales down higher frequency details, the fBm fusion equation would be given by,

$$\rho' = \sum_{i=1, \mu'_i \sim \mathcal{D}_i}^{k} a_\mu^i \mu'_i \tag{3.5}$$

This is possible only because we have samples at different frequencies and would not be possible if we used techniques incorporating mipmapping. We can compare this equation with Equation 1.4. Note that Equation 3.5 can be computed in an online manner, that is, as we desire more details, we only have to sample models $\mathcal{D}_i$ such that $i > k$ and update the previously produced patch. Equation 3.5 would thereby enable us to produce multi-resolution terrain patches ρ'_i as discussed in subsection 3.3.2

We further deal with step-size resizing in our trained diffusion models to improve our framework adaptivity. Given that we have diffusion models trained on T steps, while sampling we can reduce the number of steps to $K \leq T$ [69]. For this, K linearly spaced integers between 1 and T can be used as the input to the diffusion model. This improves the sampling speed of our model by decreasing the time to sample linearly with K. Alas, this comes at the cost of quality. This adds as another use-case for adaptive terrain sampling, which can adjust to computation constraints. An illustration and discussion on this is provided in subsection 3.3.2.

Finally, we use our enhancement module to add details to the generated terrain patch ρ'. Specifically, the super-resolution model $\mathcal{S}$ takes the unrefined patch ρ' and adds details to produce $\rho'_{SR} = \mathcal{S}(\rho')$.

3.2.6 Kernel Blending

One of the last challenges in producing infinitely large terrains is a smooth transition among the patches ρ'_{SR_i} we generated in the previous steps. Simply placing the tiles produced in a grid gives rise to discontinuities along the patch boundary, as illustrated in Figure 3.7 (a) highlighted in red. This is because the terrain tiles are produced independently and their edges do not line up with each other. On the other hand, Perlin noise function is inherently continuous in $\mathbb{R}^2$ because it smoothly interpolates

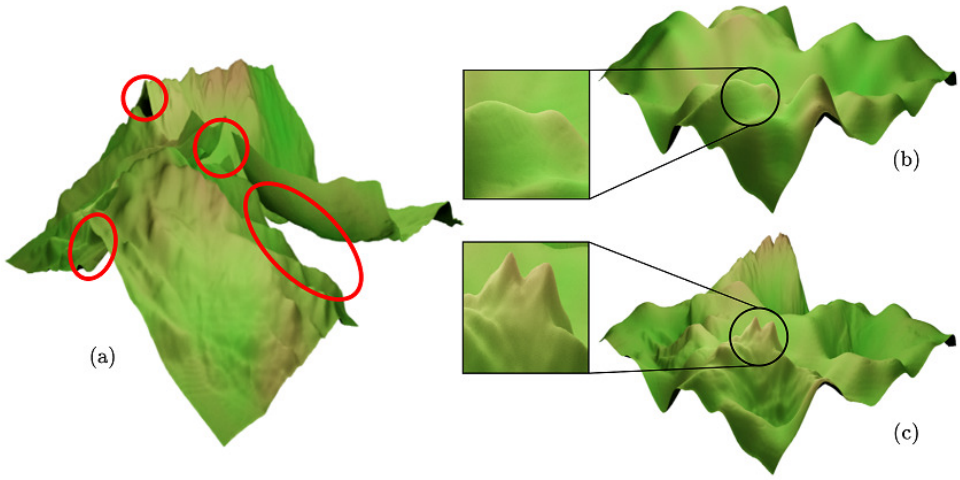


Figure 3.7: (a) Tiled terrain patches (b) Fractal Perlin noise (c) Kernel blended terrain.

between pseudo-random gradients at fixed intervals at tile vertices. Whereas, as we can observe in Figure 3.7 (b), Perlin noise does not possess real terrain features and hence its divergence from the distribution of terrain is high. We propose to take the best of both, a learnt distribution from diffusion models (Figure 3.7 (a)) and the infinite continuity of Perlin noise (Figure 3.7 (b)) by blending them using a kernel to produce infinite terrains with learnt details as illustrated in Figure 3.7 (c). The difference in details are highlighted in comparison of (b) and (c), with (b) showing a much smoother surface lacking real world detail. We do not observe boundary artifacts in (c) which were present in (a).

Let g be a 1-D kernel and $t \in [0, 1]$ represent the domain of the kernel such that $g(t) \in [0, 1]$ is the range. We want the kernel to satisfy conditions (1) $g'(t = 0) = 0$ and (2) $g'(t = 1) = 0$ for the continuity along the end-points given that we will tile this kernel, (3) $g(t = 0) = 0$ and (4) $g(t = 1) = 0$ to make sure that fractal Perlin noise is dominant at the tile edges to provide continuity and (5) $g(t = \frac{1}{2}) = 1$ such that the sampled terrain is dominant where continuity is not a concern. We require an order four polynomial to satisfy the five conditions. Let $g(t) = \sum_{i=0}^{4} a_i t^i$ be the template polynomial. Upon simplifying with constraints (1) to (5), we get the linear system of equations,

$$\begin{bmatrix} 1 & 1 & 1 \\ 4 & 3 & 2 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} a_4 \\ a_3 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 16 \end{bmatrix} \tag{3.6}$$

The solution of this gives us the kernel $g(t) = 16t^4 - 32t^2 + 16t^2$, the corresponding graph is plotted in Figure 3.8. This kernel resembles a Gaussian but upon experiments with a Gaussian kernel, we observed edge artifacts due to the tails of Gaussian being non-zero. We take the outer product of the kernel with itself which gives us the 2-D kernel $G = g \otimes g$. We blend ρ'_{SR} and fractal Perlin noise fpn for the desired domain as,

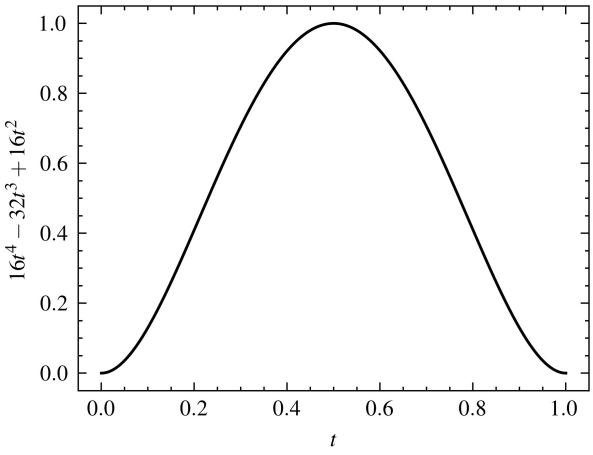


Figure 3.8: The 1-dimensional derived kernel g.

$$G \circ \rho'_{SR} + \lambda(1 - G) \circ \text{fpn} \tag{3.7}$$

where $\circ$ denotes the Hadamard product and λ is a scaling hyper-parameter that we best found to work in the range $[0.75, 1]$. An example of the resulting terrain is illustrated in Figure 3.7 (c), where Figure 3.7 (a) corresponds to ρ'_{SR} and Figure 3.7 (b) corresponds to fpn.

3.3 Experiments and Results

3.3.1 Dataset

We use the same dataset as [3, 47, 68] for fair comparison with state-of-the-art terrain generation and terrain super-resolution. This dataset is publicly available and consists of high resolution DEMs of 2m spatial resolution from the regions of Pyrenees [32] and Tyrol [19], which cover an area of 643 km^2 and 304 km^2 respectively. We divide the DEMs into tiles of 256×256 for processing in the pipeline.

3.3.2 Diffusion Model, Adaptive Sampling & Fusion

Implementation Details: We train our diffusion models with number of steps $K = 1000$. We use a batch size of 16 along with a cosine β scheduler. We learn the covariance and keep the learning rate fixed at 10^{-4} optimizing with the Adam optimizer [42]. We train for 2^{16} iterations on Nvidia GeForce RTX 2080 Ti. We keep these parameters fixed for all the frequencies. We compare our proposed diffusion based model to a set of baselines. The first baseline is the improved Perlin Noise [76] which we implemented ourselves. We set the number of octaves to 3. The second baseline is a GAN model [20] for which we use the implementation of deep convolutional GAN provided by [58]. Our batch size

Table 3.1: Comparison of Fractal Perlin [76], GAN [58], VAE+GAN (chapter 2) and the proposed diffusion based modeling using the FID ↓ metric for terrain generation.

	Fractal Perlin	GAN	VAE+GAN	Diffusion
FID ↓	335.851	204.365	119.124	**54.444**

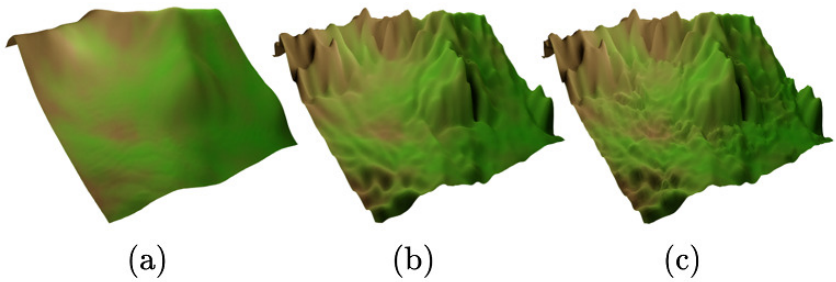


Figure 3.9: An illustration our fusion strategy (following Equation 3.5) with $a_\mu = 0.5$. (a), (b) and (c) correspond to $k = 1, 2$ and 3.

was set to 64, learning rate 2×10^{-4}, latent dimension 100, trained for 200 epochs on Nvidia GeForce RTX 2080 Ti and optimized using the Adam optimizer [42]. Our final baseline is a model which is composed of a VAE and a GAN (chapter 2). This model was trained on the same dataset as ours. For fair comparison, we set $N = 1$ for diffusion models.

Results: We generate samples from various baseline models (explained before) and compare it to the ground truth dataset using the FID metric [28], that is a metric used to compare two distriutions. The results given in Table 3.1 show that diffusion models outperform other methods in terrain generation.

In terms of qualitative understanding of adaptive terrain generation, Figure 3.10 displays terrains rendered at varying values of diffusion sampling steps (K) in which we can observe a slight drop in the finer details as we decrease K. Nevertheless, this decrease in quality can be acceptable as it yields faster generation, with $K = 1000$ taking 35s and $K = 10$ taking 5s on an Intel Xeon CPU as sampling time reduces linearly with K in diffusion models. Since terrain data is much more unstructured compared to natural images or facial data, slight deviation from the actual distribution are not caught visually, hence reducing K to improve sampling speed would not be as detrimental as it would be with other domains.

In regard to qualitative evaluation of fusion multiple patches generated by respective learnt diffusion models for different spatial frequencies is shown in Figure 3.9, where we can observe the finer details increase with an increase in k (following Equation 3.5).

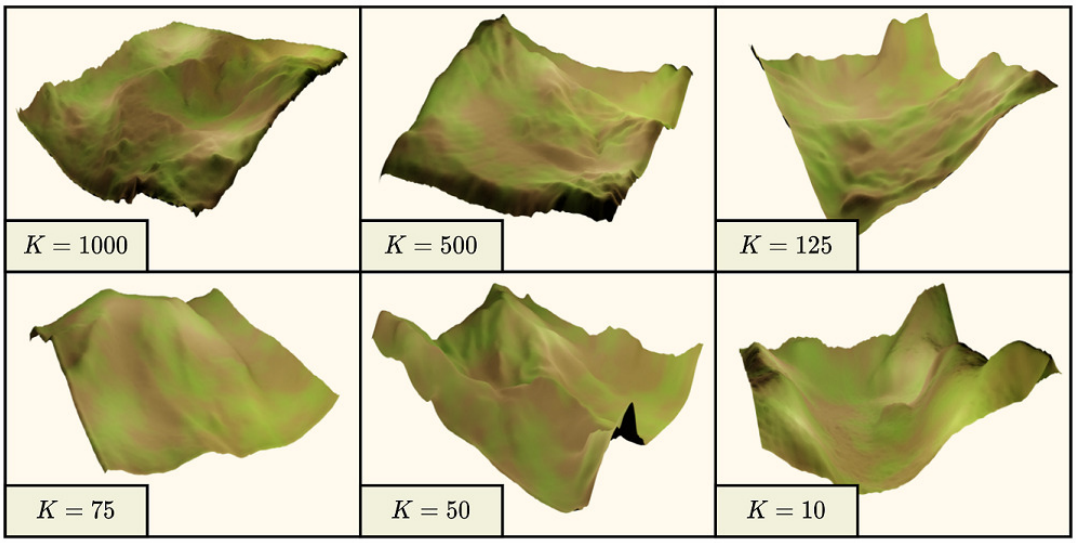


Figure 3.10: Terrains sampled with diffusion model by varying the number of sampling steps K for $N = 1$.

3.3.3 Super-Resolution model

Implementation Details: The architectural diagram for our proposed method is given in Figure 3.6, corresponding to which we found the values of $I = 8$ and $J = 16$ to work best for us. We used average pooling in our pooling layer in the process to get the attention weights. We used Adam optimizer [42] with a learning rate (LR) of 10^{-4} and a step LR scheduler reducing the LR by 5% every 2 epochs. We train for 256 epochs on Nvidia GeForce RTX 2080 Ti.

Results: Figure 3.11 illustrate qualitative result of terrain enhancement, where we can observe that (a) is the generated terrain (from fusion of multi spatial frequency patches) is smoothed out whereas fine details like sharper edges are present in enhanced terrain obtained by employing terrain super-resolution as shown in (b).

In terms of quantitative evaluation, we compare the RMSE and PSNR of our model with bicubic upsampling and two baselines models, all of which were tested on the same dataset as ours. The first baseline model FCND [3] proposed a multi-scale architecture with the possibility of combination with aerial-imagery. We use their method with just depth for fair comparison. The second baseline model [47] proposed a feedback neural network based architecture DSRFB along with extention with post-processing DSRFO. Table 3.2 compares our method with the baselines using RMSE and PSNR and establishes a new state-of-the-art in 8x terrain super-resolution. The results are reported for the test set. DSRFB should be compared with TRCAN because they are tested without post-processing and DSRFO with TRCAN+ with post-processing. TRCAN+ used a stride of 16.

We further experiment with the effect of stride on the RMSE. Figure 3.12 illustrates that there is an approximately linear relationship between the two. This can be seen as an ensemble where the variance

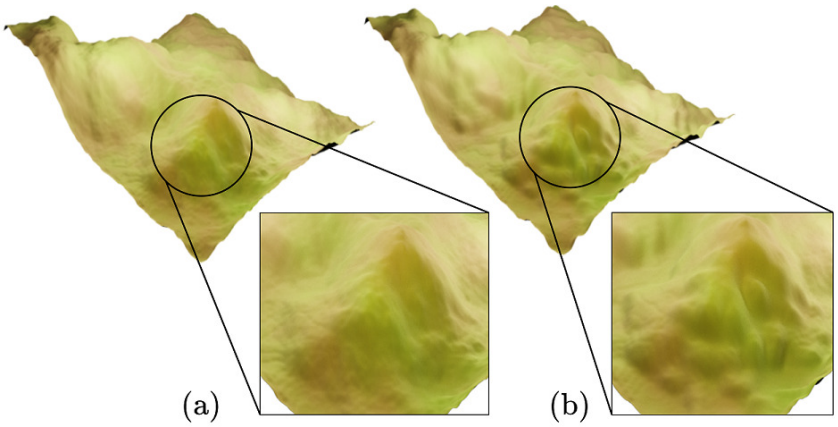


Figure 3.11: Terrain enhancement where (a) is enhanced to (b).

Table 3.2: Comparison of Bicubic Upsampling, FCND [3], DSRFB/DSRFO [47] and our proposed TRCAN/TRCAN+ using RMSE (in meter, ↓) / PSNR (in dB, ↑) for 8x terrain super-resolution.

Region	Bicubic	FCND	DSRFB	DSRFO	TRCAN	TRCAN+
Bassiero	1.406/60.5	1.146/62.261	1.091/62.687	1.083/62.752	**1.086/62.728**	**1.077/62.807**
Forcanada	1.632/58.6	1.326/60.383	1.270/60.761	1.259/60.837	**1.260/60.828**	**1.248/60.909**
Durrenstein	1.445/59.5	0.957/63.076	0.884/63.766	0.868/63.924	**0.869/63.915**	**0.847/64.138**
Monte Magro	0.917/67.2	0.632/70.461	0.589/71.081	0.581/71.196	**0.584/71.144**	**0.574/71.293**

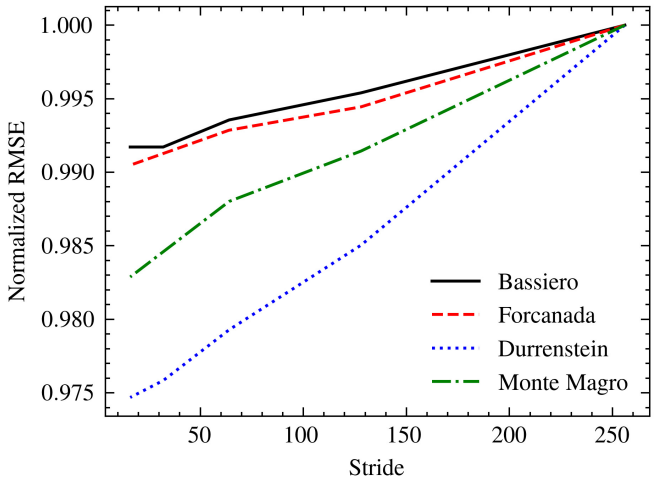


Figure 3.12: Effect of stride on the RMSE.

Table 3.3: Comparison of Fractal Perlin Noise [76], ground truth data and our proposed method based on user rating for terrain generation.

	Fractal Perlin	Ground Truth	Our
Mean	1.948	3.931	3.465
STD	1.085	1.298	1.074

in the output reduces with increasing number of samples, which is inversely proportional to the stride. The RMSE has been normalized in this plot such that the RMSE of the maximum stride is 1.0 for each region for easy visualization. The time required for post-processing increases quadratically with an increase in stride, whereas the time required for sampling tiles from the diffusion model increases linearly with the number of diffusion steps.

3.3.4 User Study

We conduct a user study with 30 domain expert participants. We show each participant 5 renders each of terrain generated via fractal Perlin Noise, our method and the ground truth dataset and ask them to rate the terrains based on realism and aesthetics out of 5. We report the mean and standard deviation (STD) of the scores in Table 3.3. We observe the expected progression in mean and STD scores with the highest scoring ground truth followed by our method and then fractal Perlin noise. We can observe that mean rating of the generated terrains obtained with Perlin noise is significantly lower as compare ratings of terrains generated with our method, which is very close to ratings given to real terrains.

3.4 Conclusion

We propose a novel framework for terrain generation introducing concepts such as frequency separation using Fourier transform, kernel blending and fBm fusion for generating patches which gives a new perspective to terrain generation in a learning based framework. Our framework is adaptive and multi-resolution for generating learning based infinite terrains procedurally which might contribute greatly to the gaming and simulation community. We test and report superior qualitative and quantitative results for diffusion based models and show their applicability for terrain generation. Along with that, we propose a state-of-the-art terrain super-resolution model with a novel post-processing technique which we employ for terrain enhancement. In the next chapter, we continue our discussion by generating objects that populate the terrain, specifically trees.

Chapter 4

Automated Tree Generation Using Grammar & Particle System

Figure 4.1: A forest rendered using the proposed model.

4.1 Introduction

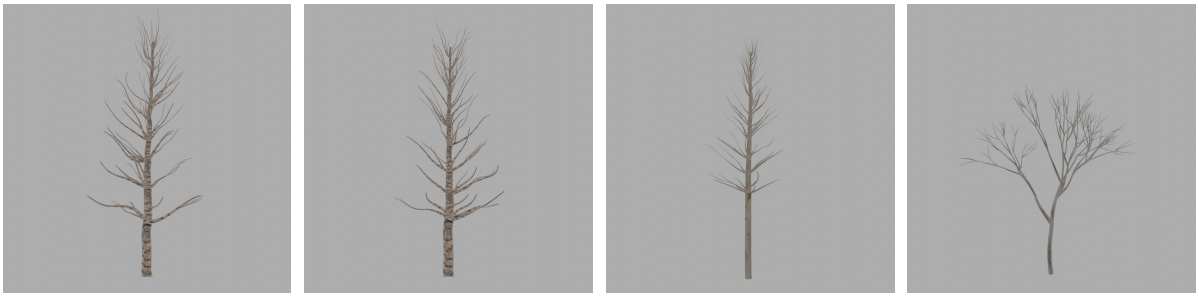
In the previous chapters, we discussed the generation of terrain, which would form the base of our virtual world. In this chapter, we continue our discussion with the generation of trees, which would be used to populate the virtual world. 3D Virtual world generation is an important goal for computer graphics applications aiming at realistic experience for gaming or virtual reality audience. Modelling realistic 3D *Flora* (plant life) is an important aspect of generating outdoor virtual scenes that are abundant

Figure 4.2: Banyan tree generated by our method.

in graphics applications. Traditionally, a set of 3D tree models designed by expert artists are repeatedly used to populate a virtual forest scene. This approach suffers from a lack of variation in 3D tree structures, as observed in real world as the same 3D tree models are being replicated. Hence, this approach fails to scale while generating virtual scenes with a large number of 3D trees, making the view monotonous and repetitive. Thus, an automated approach to generating multiple 3D trees of varying structures and appearances become important. This is a challenging task that involves modelling a large number of plant species at multiple growth stages and in different environmental conditions like lighting and wind flows.

Interestingly, trees exist as self-similar structures or fractals, where each stem is very similar to its parent stem with subtle changes in shape and width. This self-similarity can be modelled using a recursive grammar and the variation in structure is further modelled using geometric specifications. Thus, each stem can be modelled as some geometric transformation relative to its parent. Examples of such geometric transformations are scaling where child is smaller than its parent, rotation where child is present at some angle relative to its parent, etc. These observations were used by [57] in the popular L-system model to generate trees. [93] later extended earlier work to make the interpretation of grammar easier. However, they used global geometry parameters, which made it difficult to model trees like Pine that has a lot of local geometric variations.

In this chapter, we propose to extend the L-system grammar-based automation solution for 3D tree generation that overcomes the limitation of existing methods and generalize to a large variety of tree



(a) Pine (b) Pine (c) Aspen (d) Guava

Figure 4.3: Variations in grammar and geometry produce variations in trees structure.

species with varying topologies like Banyan, Mango, Pine or Palm trees. The proposed method can also be used to model related vegetation such as flowers, shrubs or grasses like bamboo. Our method uses stochastic rules and can therefore produce structural variations from the same set of rules. More importantly, the proposed method overcomes the limitations of previous methods [93] by proposing to extend the geometric variations locally instead of using global parameters. For modelling 3D foliage, we have adopted the particle system approach. This allows to model foliage unique to each tree in terms of leaf type and variations based on season, height and orientation. Subsequently, we perform texture mapping to give a realistic appearance to both stem and leaves.

Figure 4.1 shows the 3D forest scene generated by our method where we can observe the variations within multiple instances of the same species and differences between similar species consisting Pine, Aspen and Fir. It is important to note that the proposed method is intuitive and easy to understand and thus a novice user can also easily understand and edit the grammar and geometric parameters to produce trees of their specification. We also intend to release a blender plugin implementation of our method. Additionally, the majority of the existing datasets and automation based tree generation methods have focused on western tree species. We aimed at developing a library of trees focused on Indian subcontinent (e.g., Banyan Tree shown in Figure 4.2).

4.2 Tree Generation

The tree generation primarily requires modelling the structure of the tree which posses self-similar structures, albeit with large variation in size/shape within same and across various species. Additionally, the foliage is also a key component in tree modelling and poses many unique challenges like variations in size, colour, orientation and position of leaves and they also appear in large numbers as compare to stems. In this section, we discuss in detail our approach for these two key tasks.

Table 4.1: Condensed grammar for selected trees. Grammar is given as $input \rightarrow output$, with s always representing the start state and t end state where applicable. For the case of Guava, we can see that the grammar has two elements. For the second element, a can transition into aaa or aa, which can go on recursively. Each transition $a \rightarrow a$ has a geometry associated with it.

Tree	Grammar
Guava	$s \rightarrow (a); a \rightarrow (aaa/aa)$
Fir	$s \rightarrow (a); a \rightarrow (ab_1b_1b_2b_2/ab_1b_1b_2/ab_1b_1b_1); b_1 \rightarrow (t); b_2 \rightarrow (c_1c_2); c_1 \rightarrow (t); c_2 \rightarrow (t)$
Palm	$s \rightarrow (a); a \rightarrow (bb \ldots b); b \rightarrow (t)$
Bamboo	$s \rightarrow (aa \ldots a); a \rightarrow (a)$

4.2.1 Tree Structure Generation

Regarding the first task, we propose a grammar-based approach to model tree stem structures. This approach can be viewed as an *automata*, which consists of one start state $\mathcal{S}$, a set of stop states $\mathcal{T}$ and multiple transient states. Each state transition draws a stem as a spline and is controlled by a grammar $\mathcal{G}$ and the geometry $\mathcal{H}$ associated with the grammar.

Additionally, each of these transitions from one state to another state of the automata is stochastic in nature and thus can lead to one of the many next states defined as part of the grammar. This enables modelling the structural variations associated with specific tree species. Another important aspect of modelling structural variations is associated with geometrical rules defining the geometry of the stem to be drawn for that transition. Each geometric rule also has random variations within it to further increase the stochasticity within the species and also across species. This is by providing parameters such as length or angle as uniform random variables rather than a constant. Figure 4.3 shows how variations in grammar and geometry specification produce variations in trees. More specifically, trees in Figure 4.3a and Figure 4.3b share the same grammar and geometry and are from the same species, the variation produced are purely due to stochasticity in grammar and geometry. On the other hand, trees shown in Figure 4.3a and Figure 4.3c share the same grammar but have different geometries. This is shown in their similar underlying structure. Finally, trees rendered in Figure 4.3a and Figure 4.3d have different grammar as well as different geometry. In the current approach, the stems that are generated are assumed to be devoid of self-intersecting properties and is not explicitly modelled.

4.2.1.1 Grammar ($\mathcal{G}$)

Grammar describes the underlying structure of the tree. It holds structural information such as the number of stems a stem splits into, which can be stochastic. A grammar $\mathcal{G}$ is defined as a set of state transition rules, i.e., $\mathcal{G} = \{g_1, g_2, g_3, \ldots\}$ with each transition rule g_i defining the probabilities (or

likelihood) of choosing the next (output) set of states starting from current (input) state. Therefore, each element of $\mathcal{G}$ defines a stochastic input-output rule system where the input is a single state α and output can be a set of states e.g., $\{\beta, \gamma, \delta, \ldots\}$. This means that a single input state can transition into multiple output states. This is necessary for modelling trees and can be thought of as a single stem splitting into a set of multiple child stems in a stochastic manner.

Interestingly, during such transitions from input to output states, each transition $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ has a different set of geometries associated with them. Nevertheless, even if input and output states are the same, the stems will still be different due to stochasticity in the geometrical parameters as explained below. Some examples of tree specific grammars are provided in table 4.1.

Another important parameter associated with the expansion of grammar is the maximum branching depth $\mathcal{D}$. It can also be thought of as the maximum recursion depth. More specifically, it controls how many levels of branching the tree can have. Thus, it is used to control the size of the tree and hence consequently limits the resource usage of the computer by not letting the code run indefinitely in case the stop state is not encountered.

4.2.1.2 Geometry ($\mathcal{H}$)

Each tree has a set of geometric rules where each rule $h_i \in \mathcal{H}$ is associated with a every single state transition in the grammar $\mathcal{G}$. Each of these geometric rules has intuitive and simple parameters listed here:

- **Base Angle** is specified as 3 uniform random variables representing the 3 Euler angles. The range of these variables is specified by the user. The angle gives the relative change of orientation of the child stem with respect to its parent which is further modified to give the true angle.

- **Reduce angle** is used to reduce the range of angles a stem can make from its parent. It is observed that the stems that are higher up in a tree tend to make smaller angles from their parent stem than those lower. This may be attributed to external factors such as gravity which over time increases the angle between parent and child stem. It is specified as a vector containing the reduction for the 3 Euler angles. Reduce angle r_a modifies the base angle b_a to give the true angle t_a as a function of the current branching level c, given by relation $t_a = b_a \times r_a^c$.

- **Base length** of the stem is specified as a uniform random variable. This gives a relative measure of the length of the stem associated with the state transition. The true length of the stem is dependent on the base length as given below.

- **Reduce length** is specified as a scalar and it modifies the base length. True length of the stem t_l, is a function of the base length b_l, reduce length r_l and the current branching level c given by the relation $t_l = b_l \times r_l^c$. This gives an approximation of stem length as stem length reduces at higher branching levels of the tree.

(a) timestep 1 (b) timestep 2 (c) timestep 3 (d) timestep 4

Figure 4.4: The growth stages of trees. This is achieved by controlling the maximum branching depth of the tree.

- **Reduce width begin** (r_{wb}) and **reduce width end** (r_{we}) are specified as scalars. They control the widths of the endpoints of the stem and interpolate the intermediate control point widths based on the endpoint widths. Let w be the width of the adjoining endpoint of the parent stem, then the widths of the endpoints of the child stem will be $w \times r_{wb}$ and $w \times r_{we}$. Additional global parameter $\mathcal{W}$ represents the initial width of the base stem i.e., the trunk of the tree. It is a uniform random variable between ranges provided by the user. This rule assumes that the width of the child stem will always be less than or equal to its parent stem. This is true in most situations and should be a good approximation for modelling trees.

- **Curve** is a uniform random variable that specifies the curviness of the stem. Noise functions like Perlin noise can be used. The noise vector displaces the control points of the spline by an amount proportional to the curve parameter. This helps make the tree more realistic by adding noise at the structural level. Models using cylinders instead of splines for drawing stems cannot use this parameter.

4.2.2 Populating Leaves

The key challenge associated with modelling leaves is dealing with a large system of geometry. To tackle this issue, particle systems [79] have been a popular approach in the past. Particles are pieces of geometry emitted from mesh objects, typically in the thousands. Each particle can be a point of light or a mesh, and be joined or dynamic. They may react to many different influences and forces, and have the notion of a lifespan. Particle systems once created are a placeholder for the particle geometry to be placed.

We have adopted the particle system approach to model foliage unique to each tree in terms of leaf type and variations seen based on season, height and orientation. In this approach, we use hair type particles [1] which are a subset of regular particles. Hair systems form curves that can represent

(a) Green leaves (b) Red leaves (c) Yellow leaves

Figure 4.5: The variation in leaves color across seasons for maple tree.

Figure 4.6: Different stages of tree foliage across seasons.

hair and fur in addition to leaves. The simulation and modelling of human hair is a process whose computational complexity is very large, this is due to the large number of factors that must be calculated to give a realistic appearance. Generally, the method used in the film industry to simulate hair is based on particle handling graphics. In this chapter, a simpler approximation of modelling hair type particles provided in Blender [94] is used. This approach towards modelling hair type particles is a common one in the field of computer graphics.

We create a ParticleSystem in the Blender, an object parameterized to control the appearance and behaviour of individual Particle objects over time. Particles, which are born from a ParticleEmitter, have a position and type. The primary approach we take to affect the visual output is to include maximum and minimum attributes. For every variable with a maximum and minimum input, the actual value for that variable on the particle will be randomly assigned to be between the maximum and minimum input and stay statically at that value for the entire life of the particle.

When we transition from an input state to output states, then each transition has a different set of geometries associated with them. With each execution of expanding the grammar, leaves are generated as particle system for the spline. The particles system takes as input the appropriate leaf texture and distributes the leaves onto the spline that is in consideration.

Leaves are represented as planes. Leaf texture is specified as an image of the leaf. Different species have different vernacular patterns that can be specified most accurately using images of real leaves. Colour variations are observed that the stems that are higher up in a tree tend to dry out faster than the ones below. Hence the provision for multiple textures to be supplied to the tree is available. The lighter textures are seen on the higher branches and vice versa. This is achieved by calling upon the

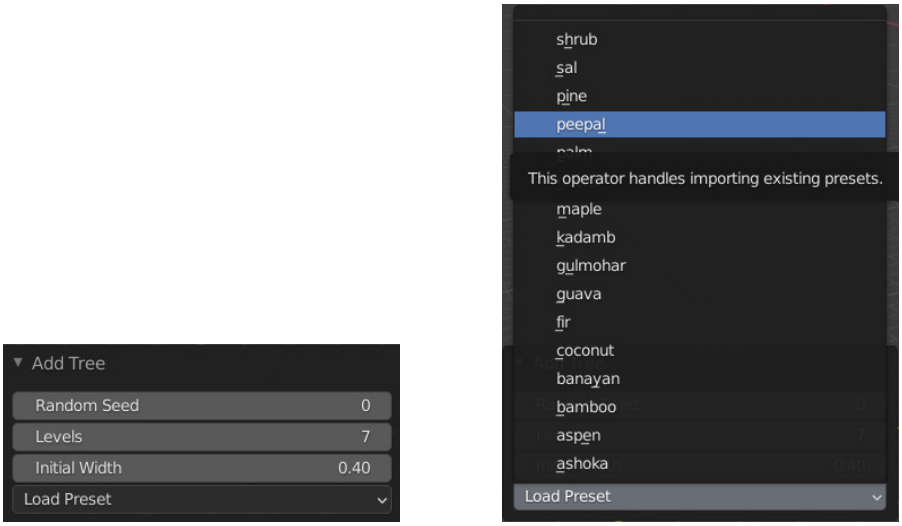


Figure 4.7: Snapshot of our Blender add-on for easy and intuitive user editing.

lighter textures for the particle systems on the top branches, a mix of lighter and darker textures on the intermediate branches and darker textures on the lower branches.

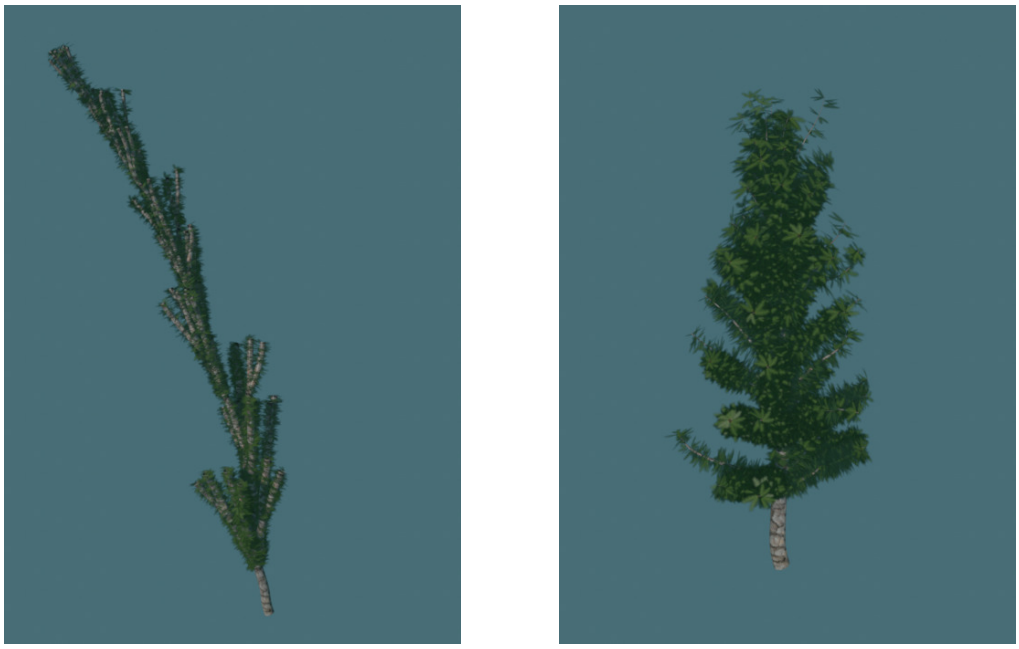
The leaves are placed at different rotation angles within a user set range. A randomness value ensures that different values are chosen within that range. The leaves are placed along the tangent to the spline. It is observed that the stems that are higher up in a tree tend to have smaller newer leaves whereas the lower branches have larger older leaves. Hence, the leaves are scaled to different sizes within a user set range. A randomness value ensures that different values are chosen within that range.

4.3 Results

We were able to generate trees of high diversity using the proposed method. This includes various western tree species (like Aspen, Fir, Maple, Pine, etc.) as well as Indian tree species (like Tulsi, Banyan, Mango, Neem, Gulmohar, Kadamb, Sal, Ashoka, Peepal, etc.) that are shown in Figures 4.9 and 4.10. Further illustrations and video are available at our website[1,2]. The code including multiple instances of various tree species as listed in Table 4.2 has been made available to the public[3].

Additionally, our method allows adding new species or updating the grammar for existing species with minimal user editing. The code for our method as a Blender package, an user-friendly interface based automated tree generation, as shown in Figure 4.7 is available at our website.

[1] https://cvit.iiit.ac.in/research/projects/cvit-projects/automated-tree-generation-using-grammar-particle-system/
[2] `https://3dcomputervision.github.io/publications/tree_generation_icvgip`
[3] `https://github.com/aryamaanjain/tree_generation_grammar`



(a) Result of Sun et al. [93]

(b) Result of our method

Figure 4.8: Qualitative comparison with [93]. (a) Keeping single global parameters for all kinds of curves [93] fails in the case of pine. (b) The proposed model specify separate local parameters for different types of curves to successfully generate a realistic Pine tree.

Table 4.2: A list of tree species and number of trees for each species in our generated dataset.

Tree	Sample	Tree	Sample
Ashoka	1000	Lemon	1000
Aspen	1000	Mango	500
Bakul	800	Maple	500
Bamboo	2000	Neem	1500
Banyan	400	Palm	2000
Coconut	1500	Peepal	400
Fir	1000	Pine	1000
Guava	1500	Sal	1500
Gulmohar	500	Shrub	800
Kadamb	1000	Tulsi	800

Our method can produce good variation in tree structures, within and across species, due to the stochastic nature of grammar and geometries as shown in Figure 4.3. In addition, our method can also be used to generate the growth pattern of a given tree species, as shown in Figure 4.4.

In regard to comparison with other relevant state-of-the-art methods, we were able to overcome shortcomings of most relevant work in [93] by proposing the usage of local rather than global parameters, which allowed us to model a much wider range of species. This is demonstrated by generating a Pine tree as shown in Figure 4.8 where our method generates a more realistic tree as compared to the method from [93].

Regarding modelling of leaves, Figure 4.5 shows variations across seasons as depicted with colours of leaves in the maple tree. Similarly, variation in leaf appearance, size and density across seasons is shown in Figure 4.6. Video results in our website shows the movement of leaves and trunks owing to external factors like strong wind currents which were produced by attaching armatures to the leaves and trunk and then animating it. We can also model additional geometrical objects like fruits, flowers by using the Particle System based approach, as shown for Mango Tree in Figure 4.10(d). Furthermore, we illustrate sample forests generated with our framework in Figure 4.11, Figure 4.12 and Figure 4.13.

The time to produce a single tree using our method ranged between approximately 10ms for trees with simple open structures like Sal to approximately 3000ms for complex and dense trees like Banayan.

4.4 Conclusion & Future Work

We propose a novel grammar based approach for generating wide variety of tree species. The proposed grammar is easy to understand and can be edited by a novice user which overcomes the complexities of parametric models. The proposed method also gives higher control to the user by setting the geometries locally. Our method is scalable due to its inherent stochasticity to produce structurally varying trees.

Some straightforward extensions of this model can be to incorporate the features such as those proposed in [98] like pruning, wind sway, vertical attraction and tropism, degradation at range. There is also a need to propose a good metric for formal comparison between various tree generation algorithms, which currently does not exist. Further its scalability for generation of vast forest stretches may need some optimisation. Learning-based approaches such as inverse procedural modelling can be integrated with this model to regress to real-world trees. Up till here, we have discussed the generation of the elements of our virtual world. Up ahead in chapter 5, we would extend those concepts and further discuss effective ways of rendering them.

(a) Aspen

(b) Bamboo

(c) Fir

(d) Guava

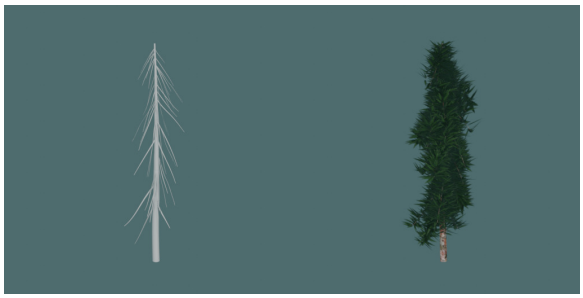
(e) Maple

(f) Palm

(g) Pine

(h) Shrub

Figure 4.9: A subset of trees found in the western world created using the proposed model.



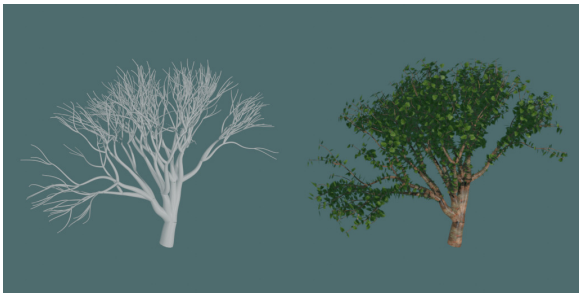
(a) Ashoka

(b) Banyan

(c) Kadamb

(d) Mango

(e) Neem



(f) Peepal

(g) Sal

(h) Tulsi

Figure 4.10: A subset of trees found in the tropical Indian subcontinent generated using the proposed model.

(a) Forest without leaves

(b) Forest with leaves

Figure 4.11: Render of forest generated with our framework with and without leaves.

Figure 4.12: Render of a generated dense forest.

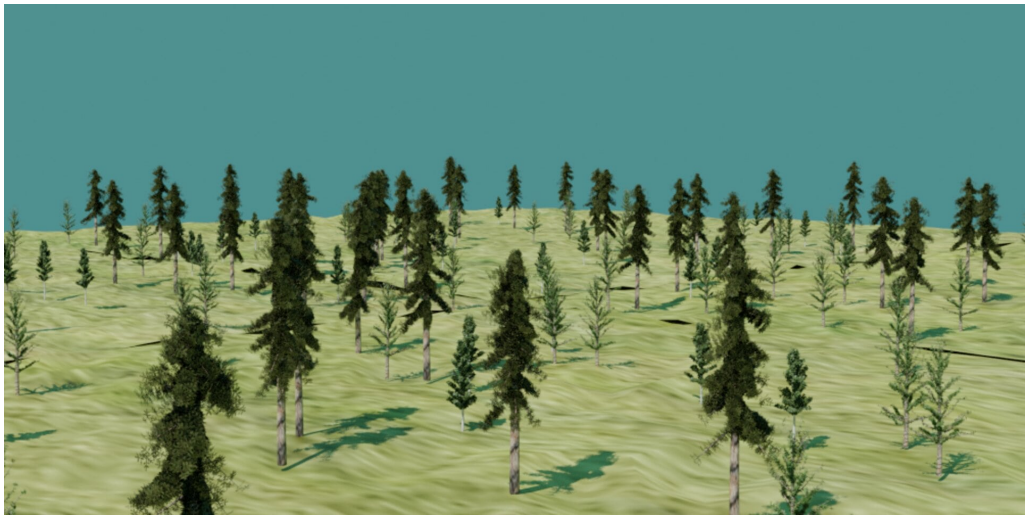


Figure 4.13: Render of a generated sparse forest.

Chapter 5

Real-Time Terrain Generation and Rendering

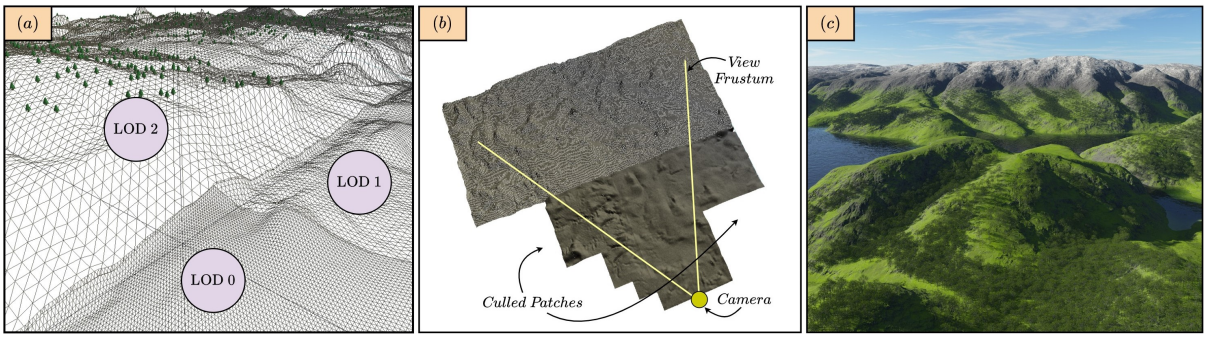


Figure 5.1: (a) We outline an approach for generating infinite terrain (learnt from real-world DEM), while incorporating level of detailing within a learning-based framework. (b) To efficiently manage the generated terrain data, a quad-tree structure is employed, which facilitates operations such as view frustum culling. (c) The entire process is executed in real-time, with the terrain being rendered simultaneously.

5.1 Introduction

In the previous chapters, we discussed ways to generate terrain and trees. This chapter culminates in extending the idea of terrain generation, populating them with trees and proposing an effective algorithm to render them.

Virtual world creation is a popular use-case for modern multi-media applications such as gaming, animation, augmented reality platforms, etc. 3D Terrain modelling is at the core of generating large-scale realistic & immersive virtual terrains, apart from populating them with trees, rocks and other objects (as shown in Figure 5.1). Synthesizing, capturing, and analyzing highly complex 3D terrain structures was traditionally pursued by Geographic Information System (GIS) community for several

critical applications such as river/flood modelling, disaster mitigation planning, and flight simulation where high fidelity to real-world data was emphasized. Digital Elevation Models (DEMs), which are raster-based representations of the terrain's elevation values, are typically used for 3D terrains. Terrain generation and rendering are the two key aspects of creating virtual terrains.

3D Terrain generation is a critical component of 3D virtual world generation, as it provides the foundational structure for the environment. Over time, 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. Traditional techniques for 3D terrain generation involve procedural generation, which relies on mathematical algorithms to generate landscapes [75, 24]. 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 [36]. In the context of rendering terrains, methods [70] use varying levels of detail to create real-time visualizations, particularly for gaming applications. Interestingly, terrain generation and rendering were typically attempted separately. Often, terrains were generated offline, followed by online rendering or even crude online terrain generation algorithms like Perlin noise were used, which did not approximate real-world data very well [61].

In this paper, we present a learning-based framework for real-time 3D terrain generation and simultaneous rendering. Our framework can generate and render infinite worlds in real time with level of detailing (LODing) that is learned from real-world data using deep learning methods. Our generative module can create terrain conditioned on their neighbourhood, while our enhancement module can refine the terrain based on quad-tree-based LODing with global-local context. Our seamless real-time rendering algorithm is compatible with both the generative and enhancement modules, providing a realistic and unique user experience.

In summary, our framework makes significant contributions to the field of terrain generation and rendering. It includes a generative module for infinite realistic terrain generation, an enhancement module for quad-tree-based LODing, and a seamless real-time rendering algorithm that is compatible with both the generative and enhancement modules. We perform an elaborate quantitative and qualitative evaluation of the proposed framework on real-world data. We plan to release our code and learned model parameters to the community to further encourage research in this area.

5.2 Method

We propose a real-time terrain generation and rendering framework. An overview of our framework is given in Figure 5.2. We start with an initial DEM, which can either be taken from an existing dataset or generated via a terrain patch generation algorithm like [23, 68]. We then employ a terrain completion module (TCM) (Figure 5.2b) to extend the initial DEM and enable the generation of infinite worlds. TCM is discussed in detail in subsection 5.2.1. TCM generates terrain at the coarsest scale and needs

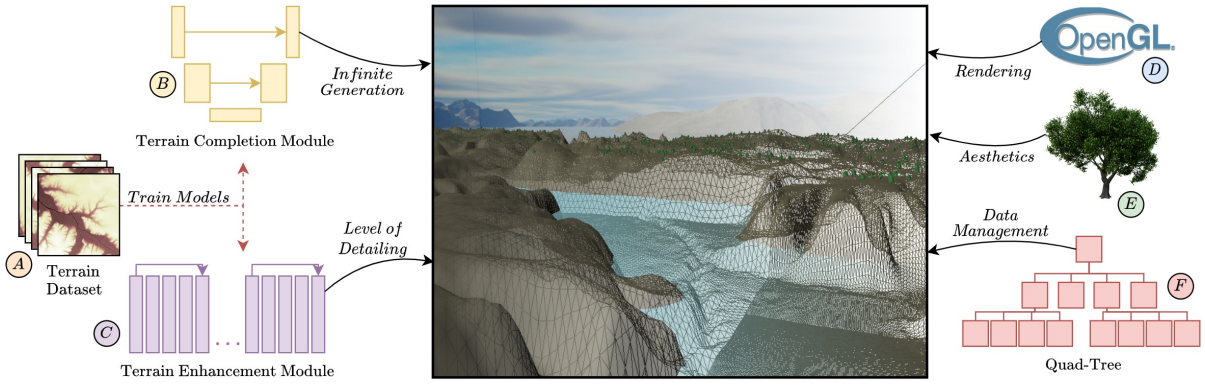


Figure 5.2: Overview of our framework with learnt terrain completion and enhancement modules. A quad-tree data structure is used for data management along with rendering in OpenGL.

to be enhanced for Level of Detailing (LODing) based on factors such as viewing position, direction and roughness of the terrain. We propose a novel quad-tree-based terrain enhancement module (TEM) for LODing of the terrain (Figure 5.2c), which is elaborated in subsection 5.2.2. Finally, we propose a terrain rendering algorithm (subsection 5.2.3) which is seamlessly integrated with the terrain completion and enhancement modules.

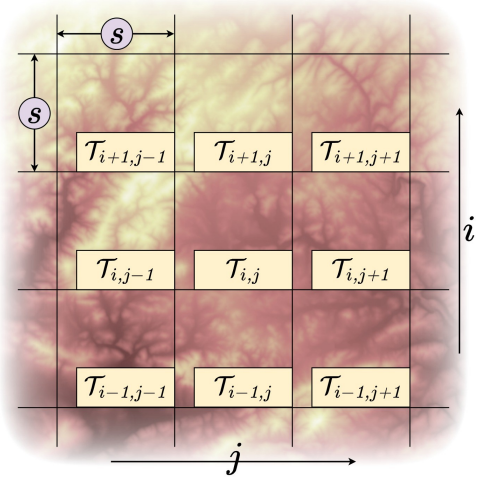


Figure 5.3: Illustration of the formulation and notations.

The notations used in the proceeding sections are illustrated in Figure 5.3. Denoting an infinite terrain grid as $\mathcal{T}$, let the individual patches of the grid $\mathcal{T}$ be indexable as $\mathcal{T}_{i,j}$ where $i, j \in \mathbb{Z}$ represent the row and column indices of the terrain grid. $\mathcal{T}_{i,j}$ is a fixed size DEM square patch of dimension $s \times s$. Each indexing operation gives us a terrain patch $\mathcal{T}_{i,j} \in \mathcal{P}$, where $\mathcal{P}$ represents the set of all the terrain patches.

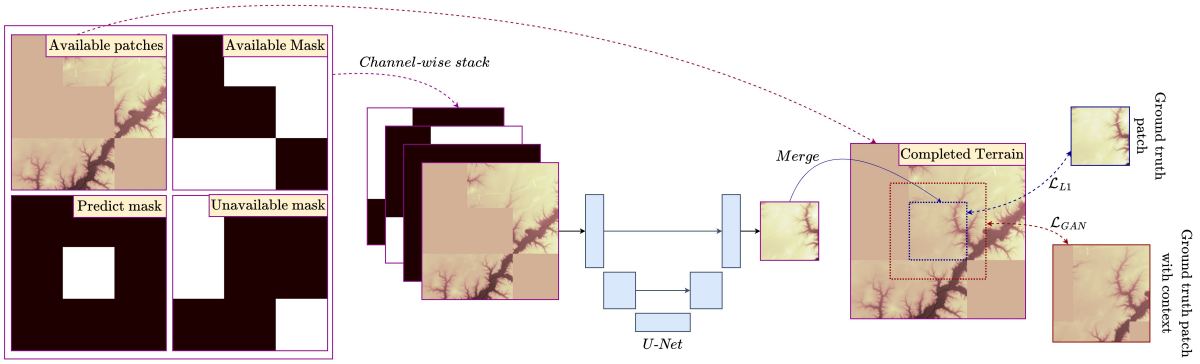


Figure 5.4: Overview of the terrain completion module.

5.2.1 Terrain Completion Module

We pose the problem of terrain completion in a discretized and structured setting. Given a subset of the 8-neighbourhood patches of a terrain patch as input, we aim to generate the centre patch. More formally, to generate patch $\mathcal{T}_{i,j}$, the input to the model would be $\{\mathcal{T}_{i+p,j+q} \mid (p,q) \in \{-1 \dots 1\}^2 \wedge (p,q) \neq (0,0) \wedge I(\mathcal{T}_{i+p,j+q})\}$ where $I(\mathcal{T}_{i+p,j+q})$ is true if $\mathcal{T}_{i+p,j+q}$ is available. Solving this problem will allow us to generate infinite terrain as we would be able to extend the given or previously generated terrain patches in an unstructured manner. This model would be capable of unconditional generation in case of no available context patches, outpainting in case of context from a single direction and inpainting in other cases. Figure 5.4 illustrates the overview of the proposed terrain completion module.

The input to our model during training is generated from the ground truth DEM. We mask the 8-neighbour patches of the centre terrain patch in the ground truth DEM independently and randomly. This generates a scenario that we might encounter during inference, where the terrain grid $\mathcal{T}$ would be filled based on the trajectory of the user, producing arbitrary neighbourhoods. Furthermore, we stack the input with 3 masks along the channel corresponding to 1) predict mask: 1 for the centre patch and 0 elsewhere (constant for all cases), corresponding to the patch we want to predict 2) available mask: 1 where the neighbouring patches are available and 0 elsewhere, corresponding to the neighbouring patches which are available and 3) unavailable mask: which corresponds to the neighbouring patches which are not available. We use a UNet [81] architecture for the generator G in a GAN-based framework. The generator generates the patch $\mathcal{T}_{i,j}$. We then merge the generated patch with the previously available terrain patches in $\mathcal{T}$.

We use a combination of L1 and GAN losses to optimize the parameters of the network which have shown promise in the generative literature in frameworks such as Pix2Pix [35]. More specifically, let the generated patch be denoted by $\hat{y}$ and the ground truth patch by y. We define the $L1$ loss as,

$$\mathcal{L}_{L1}(G) = \|y - \hat{y}\|_1 \tag{5.1}$$

Furthermore, the discriminator D of the GAN is modelled as a CNN. Let the generated patch along with a neighbourhood context extending beyond the boundary of the patch (as shown in the red dotted box in Figure 5.4) be denoted by $\hat{y_c}$ and ground truth patch along with context be denoted by y_c. We define the GAN loss as,

$$\mathcal{L}_{GAN}(G, D) = \min_{G} \max_{D} \log(D(y_c)) + \log(1 - D(\hat{y_c})) \tag{5.2}$$

Work such as [34] has shown the importance to take *local* patches as well as *global* context for meaningful inpainting. Combining the terms, the final loss for the terrain completion module is,

$$\mathcal{L}_c = \mathcal{L}_{GAN}(G, D) + \lambda_c \mathcal{L}_{L1}(G) \tag{5.3}$$

Here, λ_c is a hyper-parameter to adjust the contribution of the two loss terms.

We show the inference stage of the module in algorithm 1. The relevance radius (R) describes the L_2 distance beyond which we do not process terrain patches. We add the result of every inference to the terrain grid $\mathcal{T}$.

ALGORITHM 1: Inference stage of TCM

Data: x, z ◁ viewer position
R ◁ relevance radius
$\mathcal{T}$ ◁ terrain grid

forall $(u, v) \mid \|(u, v) - (x, z)\|_2 \leq R \wedge \neg available(\mathcal{T}_{u,v})$ **do**
    // complete all unavailable patches in relevance area
    $np \leftarrow getNP(u, v, \mathcal{T})$ // get available 8-neighbouring patches
    $cp \leftarrow TCM(np)$ // predict the center patch given neighbours
    $\mathcal{T} \leftarrow updateGrid(\mathcal{T}, cp, u, v)$ // add patch to grid

5.2.2 Terrain Enhancement Module

Terrain enhancement is the process of adding details to coarse terrain. This can be achieved through traditional methods like running erosion simulations or more recently, using deep learning-based methods like super-resolution. We pose the problem as a super-resolution problem. 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 8x [55, 115], for terrains as in our case, the factor can be as high as 32x. This makes directly applying methods from natural image super-resolution literature to terrains non-trivial. For example, as in our case, the input DEM is 256^2 raster grid and 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 memory constraints of the GPU VRAM. While previous work [46, 47, 3, 36] have attempted to solve this

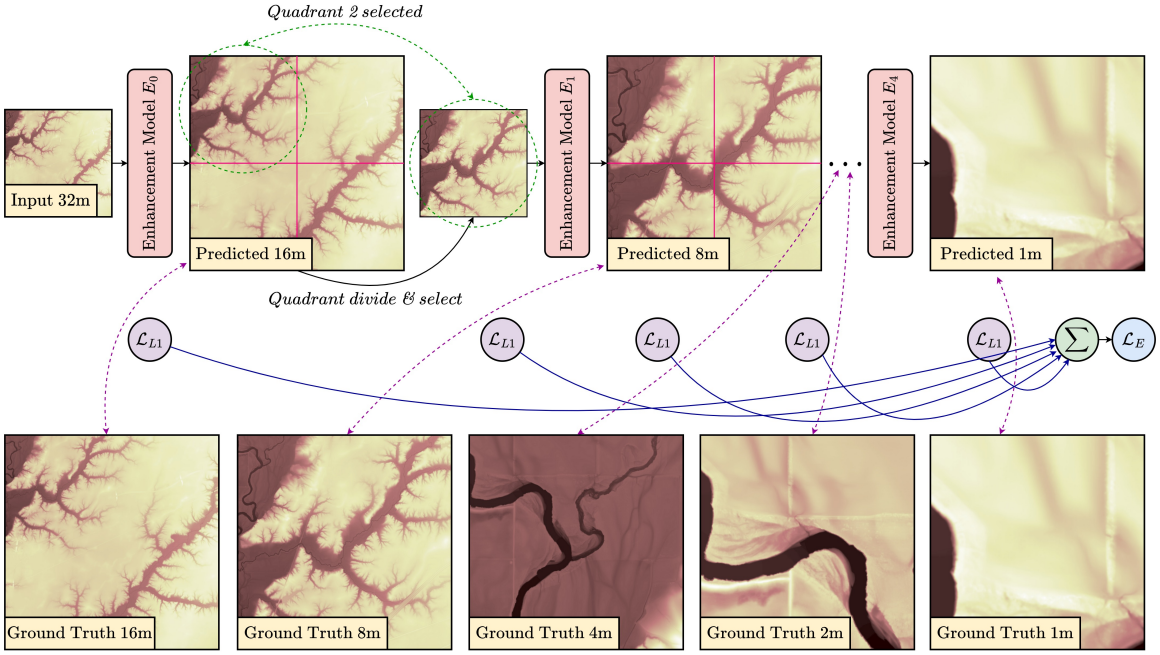


Figure 5.5: Overview of the terrain enhancement module.

problem by dividing the terrain into patches and then processing each patch individually, they have a practical limitation due to each patch being processed independently of the other, which might be reasonable locally, but have no signals from a global context. They also suffer from boundary artefacts [47] due to their tile-based processing. In order to overcome this bottleneck, we propose a quad-tree-based enhancement module which takes global and local coherence into account while processing the terrain patches.

An overview of the terrain enhancement module is illustrated in Figure 5.5. We aim to enhance a given terrain patch $T_{i,j} \in \mathcal{P}$. The terrain patches produced by the TCM are of scale 32m and we aim to enhance the patch to scales of 16m, 8m, 4m, 2m, and 1m for LODing in the rendering phase. This is similar to progressive super-resolution, but rather than processing the entire patch, we process quadrants of the patch. This makes sure that the enhancement module receives input of the same dimension at each stage unlike [48, 49, 97], where the dimensions of the input increases at successive stages limiting their scalability for higher enhancement factors. Our formulation has the added advantage of being compatible with the quad-tree-based terrain rendering algorithms which have been researched quite extensively.

We employ m models $E_{0...m}$ for enhancement (m=5 in our implementation), with model E_0 enhancing a 32m resolution DEM to 16m, E_1 from 16m to 8m and so on. We justify this design choice rather than choosing one common model by observing the ground truth DEMs at different scales in Figure 5.5, which demonstrates different features at different scales. This is another point of distinction from natural images which may have the same features at different scales due to projective imaging.

A single update step of our training algorithm is given in algorithm 2. The input patch is denoted by p_{32} where 32 denotes the terrain at 32m scale. Furthermore, algorithm 2 takes ground truth patches $p_{16}, p_8, p_4, p_2, p_1$ as input for supervision along with the models E_0, E_1, E_2, E_3, E_4 whose parameters are updated. Moreover, we found augmenting features to the input terrain helpful (discussed more in subsection 5.3.6) for enhancement given in line 5 of algorithm 2. We run the algorithm until convergence. This simple yet effective algorithm gives us the ability to perform enhancement at scale.

ALGORITHM 2: Single step of TEM training

Data: $\hat{p}_{32} \in \mathcal{P}$ ◁ LR input patch
$p_{16}, p_8, p_4, p_2, p_1$ ◁ HR ground truth patches
E_0, E_1, E_2, E_3, E_4 ◁ enhancement models

1 $scale \leftarrow (32, 16, 8, 4, 2, 1)$ // will aid in indexing
2 $loss \leftarrow 0.0$ // initialize loss to zero
3 **for** $i \leftarrow 0$ **to** 4 **do**
4 $\quad quad \leftarrow selectQuad(\hat{p}_{scale[i]})$ // select random quadrant of patch
5 $\quad quadFE \leftarrow FE(quad)$ // add features to the quadrant
6 $\quad \hat{p}_{scale[i+1]} \leftarrow E_i(quadFE)$ // forward pass
7 $\quad loss \leftarrow loss + \|\hat{p}_{scale[i+1]} - p_{scale[i+1]}\|_1$ // accumulate loss
8 $backprop(loss)$ // back-propagate
9 $Adam(E_{0..4})$ // optimize parameters with Adam

The inference of TEM is implemented as a recursive function given in algorithm 3. The $shouldEnhance()$ function is based on a number of factors including the roughness (standard deviation) of the patch, intersection with view frustum, distance from viewpoint and availability of enhancement models beyond a certain depth in the recursion. We can observe a quad-tree based inference in the algorithm where the patch is divided into four quads at each level. Although not shown in algorithm 3 for brevity, we cache the results in a quad-tree data structure for faster inference.

5.2.3 Terrain Rendering

Terrain rendering refers to the process of creating a visual representation of a three-dimensional terrain or landscape, for which we use a quad-tree-based algorithm. The rendering algorithm starts with an initial input patch which can be given from an existing DEM or generated via a terrain generation algorithm [23, 68]. We use the TCM to fill $\mathcal{T}$ with terrain patches where unavailable within a pre-defined radius from the user's viewpoint as the user moves to approximate an infinite view.

For LODing, we make use of the TEM, wherein we split the patch based on two criteria 1) the distance from the user viewpoint and 2) the surface roughness. The surface roughness is approximated as the standard deviation of the patch. We maintain the split tiles in a quad-tree data structure, where the coarsest tile is at the root and each node has 4 children of twice the resolution, corresponding to the

ALGORITHM 3: Recursive inference function of TEM

Input: P ◁ patch to render
x, z ◁ viewer position
$depth$ ◁ recursion depth
Data: E_0, E_1, E_2, E_3, E_4 ◁ enhancement models

/* Enhance based on LOD criteria */
if ¬ *shouldEnhance(P, x, z, depth)* **then**
  $loadBuffer(P)$ // send patch to GPU
  $renderPatch(P)$ // render patch
else
  /* LODing: enhance quadrants of patch recursively */
  $TEM(E_{depth}(P_{\text{ne}}), x, z, depth + 1)$ // north-east quadrant
  $TEM(E_{depth}(P_{\text{nw}}), x, z, depth + 1)$ // north-west quadrant
  $TEM(E_{depth}(P_{\text{se}}), x, z, depth + 1)$ // south-east quadrant
  $TEM(E_{depth}(P_{\text{sw}}), x, z, depth + 1)$ // south-west quadrant

north-east, north-west, south-east and south-west tiles. The formulation of our enhancement module makes the integration with the rendering module seamless.

A quad-tree-based rendering makes optimizations like view-frustum culling very efficient (as illustrated in Figure 5.1b), as we can discard the tiles that are not visible to the user for rendering via an inexpensive frustum-tile intersection check. We give an overview of the rendering algorithm in algorithm 4.

ALGORITHM 4: Terrain rendering algorithm

Data: $camera$ ◁ camera object
TCM, TEM ◁ terrain completion & enhancement modules
R ◁ relevance radius

$ip \leftarrow processUserInput()$ // get user inputs like movement
$camera \leftarrow updateCamera(ip)$ // update attributes like position
$updateShaders(camera)$ // update attributes like MVP matrices
$x, z \leftarrow camera_x, camera_z$ // get user position in terrain grid
$TCM(x, z, R)$ // complete terrain using TCM
forall $(u, v) \mid \|(u, v) - (x, z)\|_2 \leq R$ **do**
  $patch \leftarrow \mathcal{T}_{u,v}$ // extract patch to render
  $TEM(patch, x, z, 0)$ // render with LODing using TEM
$drawOtherObjects()$ // trees, water bodies, sky

Other Rendering Details:

- Our observations reveal that minor misalignments occur between tiles generated by the TCM. To rectify this, we employ an inverse distance weighting function along a small strip along the edge to align the tiles with their adjacent counterparts.

- In order to compute the normals necessary for shading, we implement a finite difference method to determine the gradients along the x and z axes. This approach is more efficient than determining the normals via cross-products. The efficacy of this method is attributed to the storage of the DEM as a 2D grid.

- To enhance the aesthetics of our model, we incorporate objects, such as trees, at scale using geometric instancing. This allows for the optimization of the rendering process by reducing the number of draw calls required.

5.3 Experiments and Results

5.3.1 Dataset

The experiments employed the USGS National Map 3DEP Downloadable Data Collection [1] 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. Subsequently, the dataset was cropped to the nearest power of two, yielding a final dataset size of 8192^2 at 1-meter resolution.

5.3.2 Evaluation Metrics

- **Fréchet Inception Distance** (FID) [27], a metric commonly used in generative models, is utilized to evaluate the quality of generated samples by comparing the feature statistics extracted from the generated samples to those of real samples using the Inception network. A lower FID score indicates higher quality generated samples. We employ the FID metric to assess the performance of the TCM model.

- **Root Mean Squared Error** (RMSE) and **Peak Signal-to-Noise Ratio** (PSNR), widely used in image and video processing tasks, are two performance metrics we use to evaluate the performance of the TEM. RMSE measures the average difference between predicted and actual values, while PSNR measures the ratio between the maximum possible power of a signal and the power of corrupting noise.

[1] `https://www.sciencebase.gov/catalog/item/543e6b86e4b0fd76af69cf4c`

- **Frames Per Second** (FPS) is a commonly used metric for measuring the performance of real-time and interactive applications, such as video games. It denotes the number of frames that can be rendered and displayed on the screen per second, with higher FPS indicating smoother and more responsive gameplay. We employ this metric to evaluate the performance of our rendering algorithm.

- **User study** is a popular research method in which participants interact with a system or product to assess its usability, user experience, and effectiveness. We conduct a *first preference experiment*, a type of user study in which participants are asked to choose their preferred option from a set of alternatives based on specific criteria, to evaluate the quality of the TCM.

5.3.3 Implementation Details

TCM is a Generative Adversarial Network (GAN) composed of a generator and a discriminator. The generator component of the 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, with 4 and 1 representing the number of input and output channels, respectively. To enhance training stability, batch normalization was incorporated into the standard UNet architecture. The discriminator component of TCM comprises a Convolutional Neural Network (CNN) followed by a Multi-layer Perceptron (MLP). The discriminator can be conceptually divided into four parts: head, body, tail, and classifier. The head converts the four input channels to a predefined number of channels N^c_{tcm}, which is set to 64. The body maintains N^c_{tcm}(=3) channels across a series of N^b_{tcm} `Conv - BatchNorm - LeakyReLU - Conv - Maxpool` layers. Finally, the tail reduces the number of channels to N^t_{tcm} (= 1) and passes flattened features to a two-layer MLP. To optimize the TCM, we employed the Adam optimizer 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 0.75 every four epochs. We trained the model using a batch size of 8 for 256 epochs. Furthermore, we set the λ_c value to 10 for our optimization equation (5.3).

Figure 5.12 and Figure 5.13 depict the generator and discriminator of the GAN-based TCM module, respectively. The generator produces a 128×128 patch with a resolution of 32m, covering an area of approximately 16 million sq km, given a context of 384×384 patches with the center being predicted. Meanwhile, the discriminator takes the predicted patch and a shorter appended context (i.e., a patch of resolution 192×192) and predicts whether the patch is real or fake.

TEM is composed of a set of convolutional neural network (CNN) models, with the number of models m set to five. Each model contains a small number of trainable parameters, specifically 47497, in order to ensure real-time performance during inference. Conceptually, 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 an input of N^f_{tem} (=4) channels, consisting of a DEM to enhance and $N^f_{tem}-1$ additional features, and outputs N^c_{tem}(=32) channels. The three additional channels in the input

include the gradient in the x direction, gradient in the z direction, and L1 norm of the gradient. The body component is composed of N^b_{tem} (=4) `Conv - ReLU` layers, which maintain N^c_{tem} channels. The tail component includes a Pixel-Shuffle layer [88] 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 a LR of 0.001 and a scheduler that reduces the LR by 0.8 every two epochs.

Figure 5.14 and Figure 5.15 illustrate the architecture diagrams for TEM and TEM$_{big}$, respectively. TEM$_{bprop}$ shares the same architecture as TEM, while TEM$_{dem}$ differs only in the input and the first convolutional layer, which processes a single channel input rather than 4 channels. We note that TEM$_{big}$ has a greater number of layers in the body than TEM and processes 64 channels in the head, body, and tail, compared to 32 in TEM.

5.3.4 Training Details

In all our experiments, we adopt an 80-10-10 train-validation-test split. The training set comprises 1392 terrain tiles, while the validation and test sets contain 174 tiles each. Each tile has a resolution of 8192×8192.

Regarding the TRCAN model, we set the number of resgroups to 5, the number of resblocks to 10, and the number of features to 48. We train the model using the Adam optimizer for L1 loss, with a learning rate (LR) of 1e-4 and a step LR scheduler that reduces the LR by 0.95 every 2 epochs. For AFND, we set the number of features to 64 and the number of steps to 2. We train this model with the Adam optimizer and a LR of 1e-4.

The models were trained on a NVIDIA GeForce RTX 2080 Ti GPU employing the PyTorch framework.

5.3.5 Terrain Completion

Quantitative evaluation: We adopt the Fréchet Inception Distance (FID) metric [27] to quantitatively evaluate the TCM and present the corresponding results in Table 5.1. Perlin noise is employed as the baseline for comparison. Our evaluation demonstrates that TCM (FID 114.376) outperforms Perlin noise (FID 495.256) in generating terrains that closely match the distribution of real-world terrains. Additionally, an ablation study is conducted to justify the choice of our parameters, as shown in Table 5.1. Our findings reveal a decreasing trend in FID as the ratio of generator to discriminator parameters increases, with TCM$_a$, TCM$_b$, TCM$_c$ and TCM arranged in increasing order of this ratio. 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 parameters such as N^c_{tcm} and N^b_{tcm}. We cap the maximum size of generator to 31 million parameters as networks beyond this size inhibit the real-time inference goal while rendering. A visual comparison of TCM and Perlin noise is presented in Figure 5.6, where it

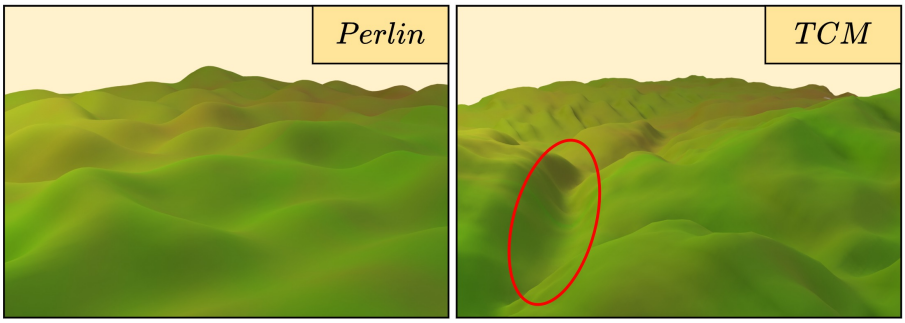


Figure 5.6: Comparison of terrain generated by Perlin Noise and TCM. The terrain generated by TCM exhibits features such as valleys (highlighted in the red circle) that are not present in the Perlin Noise-generated terrain.

Table 5.1: Quantitative evaluation of TCM using FID↓[27]. The number of learnable parameters of the Generator($_G$) and Discriminator($_D$) is denoted by S, and the number of channels in the bottleneck of U-Net is denoted by W. The best result is highlighted in orange.

Method	**FID**	**W**$_G$	**S**$_D$	**S**$_G$/**S**$_D$
Perlin[76]	495.256	NA	NA	NA
TCM$_a$	386.024	256	4806033	0.388
TCM$_b$	215.854	256	1214217	1.536
TCM$_c$	148.892	512	245762	31.334
TCM	114.376	1024	245762	126.294

can be observed that the terrain generated by Perlin noise lacks distinctive features such as valleys and ridges.

User study: For subjective evaluation, we conducted a user study involving 16 participants with varying levels of experience in terrain analysis. In order to evaluate the effectiveness of our approach, we employed a *first preference experiment* methodology. Each participant was presented with 5 pairs of terrain patches, where each pair consisted of a terrain generated using Perlin noise and TCM, randomly ordered. The users were asked to evaluate the perceived realism of the terrain structure by selecting their first preference between the two options, denoted as A and B. Figure 5.7 illustrates the five pairs of images presented to the users where options 1B, 2A, 3A, 4B and 5B correspond to the TCM generated samples and their complementary correspond to the Perlin noise generated samples. The participants expressed a clear preference for the terrain patch generated with TCM in 93.75% of the cases, highlighting the superior performance of our method. Specifically, out of 80 total responses, 75 (93.75%) favored TCM, while only 5 responses favored Perlin noise. Notably, three of the Perlin noise preferences were recorded for pair 2, while the remaining two were for pair 5. These findings suggest that TCM is a more effective method for generating realistic terrain structures compared to Perlin noise. However, it is important to acknowledge that an expert in terrain and simulation commented that "the generated terrain lacks geologically accurate fluvial dendritic patterns", which may require an explicit inclusion of geological constraints. Although this may be necessary for specific GIS use cases such as flood modelling, it may be relaxed for multimedia applications and games.

5.3.6 Terrain Enhancement

Comparison: We present a comparison of the TEM with a bicubic upsampling baseline and two existing terrain super-resolution methods, namely AFND[2][46], and TRCAN[3] (chapter 3). Our evaluation is based on RMSE and PSNR metrics, as shown in Table 5.2. AFND and TRCAN are tile-based super-resolution methods that were originally designed for enhancement factors up to 8, but we have trained them to handle factors up to 32. However, since these methods process each tile independently without global context, their performance decreases as the number of tiles increases with increasing enhancement factors. In contrast, the TEM has an image pyramid structure and is not affected by this drawback. Our experimental results demonstrate that the TEM outperforms both AFND and TRCAN for enhancement factors greater than two. Overall, our findings highlight the effectiveness of the TEM for terrain enhancement, especially for high enhancement factors.

Ablation: We report the results on ablation tests for TEM in Table 5.3. Our final model is denoted as **TEM** in the third row of the table. In $\mathbf{TEM}_{dem}$, we pass the DEM as input without any features, which leads to lower scores. In $\mathbf{TEM}_{bprop}$, we back-propagate with only the last model's loss without any direct supervision for other models. Specifically, we use the loss term $\|\hat{p}_1 - p_1\|_1$, where $\hat{p}_1$ and p_1 are the predicted patch and ground truth patches at 1m, instead of the summation of L_1 losses for all

[2] https://github.com/ashj9/AFN

[3] https://github.com/aryamaanjain/trcan

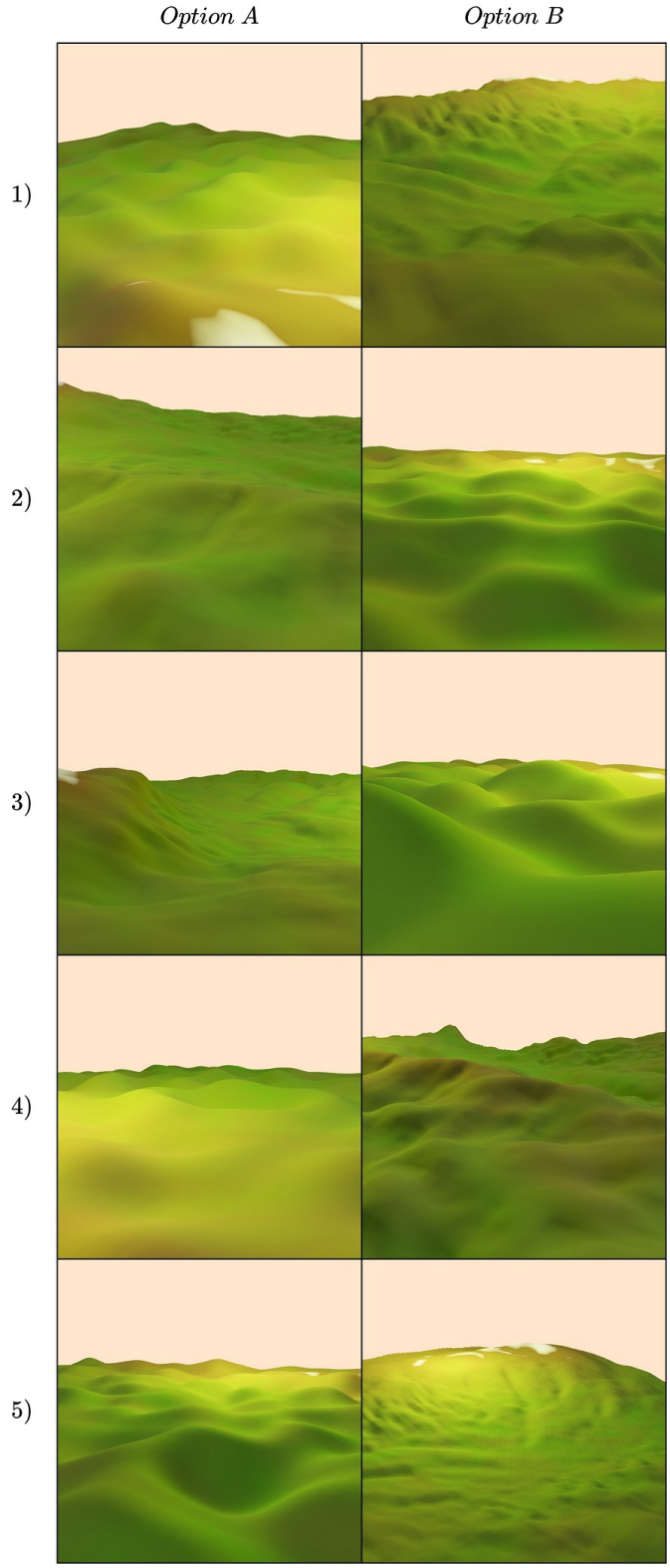


Figure 5.7: Samples used in the user study.

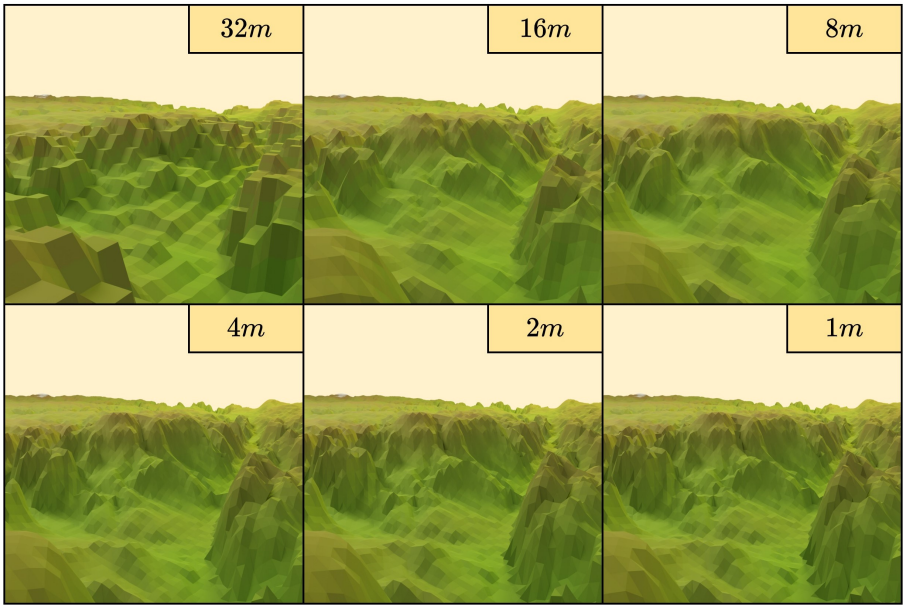


Figure 5.8: A 32m resolution input LR DEM progressively enhanced with the TEM.

Table 5.2: This table compares the performance of the TEM with other models measured using RMSE↓ (in meter) and PSNR↑ (in dB). Orange and yellow highlights indicate the best and second-best results respectively.

Method	**16m (2x)**		**8m (4x)**		**4m (8x)**		**2m (16x)**		**1m (32x)**	
	RMSE	**PSNR**	**RMSE**	**PSNR**	**RMSE**	**PSNR**	**RMSE**	**PSNR**	**RMSE**	**PSNR**
Bicubic	0.587	39.225	0.545	39.801	0.553	39.791	0.57	39.919	0.611	39.924
AFND[46]	0.451	40.983	0.459	40.662	0.489	40.438	0.505	40.314	0.584	40.08
TRCAN(chapter 3)	0.368	41.937	0.408	41.124	0.436	40.869	0.471	40.532	0.488	40.414
TEM	0.417	40.801	0.394	41.511	0.384	41.984	0.372	42.551	0.357	43.316

Table 5.3: This table presents the results of the ablation study of TEM measured using RMSE↓ (in meter) and PSNR↑ (in dB). Orange and yellow highlights indicate the best and second-best results respectively.

Method	16m (2x)		8m (4x)		4m (8x)		2m (16x)		1m (32x)	
	RMSE	**PSNR**	**RMSE**	**PSNR**	**RMSE**	**PSNR**	**RMSE**	**PSNR**	**RMSE**	**PSNR**
TEM_{dem}	0.443	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
TEM	0.417	40.801	0.394	41.511	0.384	41.984	0.372	42.551	0.357	43.316
TEM_{big}	0.414	40.884	0.394	41.548	0.381	42.086	0.371	42.671	0.357	43.442

resolutions as given in algorithm 2. 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 32x but also find that the performance is not the best for 32x enhancement factor. These observations further support the potential of a progressive architecture for terrain enhancement. Furthermore, we also experiment with a scaled-up version of TEM, denoted as $\textbf{TEM}_{big}$, which contains 224081 trainable parameters ($\times$ 5 models) compared to 47497 trainable parameters ($\times$ 5 models) of TEM. While TEM_{big} 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 5.8 where finer details are added progressively, with the finest details in the 1m terrain.

5.3.7 Patch Edge Errors

Patch-based methods such as AFND[46] and TRCAN[36] are known to exhibit high errors along patch boundaries, especially for high enhancement factors such as 32x. This limitation becomes more pronounced as the number of patches increases (quadratic increase with enhancement factor). We attribute this limitation to the lack of global signals in patch-based methods and propose a solution by designing the TEM as an image pyramid that takes global context into account.

Our proposed solution is illustrated in Figure 5.9, where the first column depicts the ground truth, the second and third columns show the 32x enhancement results from TEM and TRCAN (patch-based), respectively, and the last two columns display the corresponding error maps obtained by subtracting the ground truth from the enhancement. It is notable that significant errors occur along the edges of TRCAN (patch-based) enhancement, as highlighted in black, which are absent in TEM. The error maps in Figure 1 display positive errors in red, negative errors in blue, and zero errors in white.

The observed behavior leads to a degradation in the RMSE/PSNR of patch-based methods with increasing enhancement factors and an increasing number of patches. This limitation persists despite the decrease in the variance of patch elevations at higher resolutions. Overall, our proposed solution

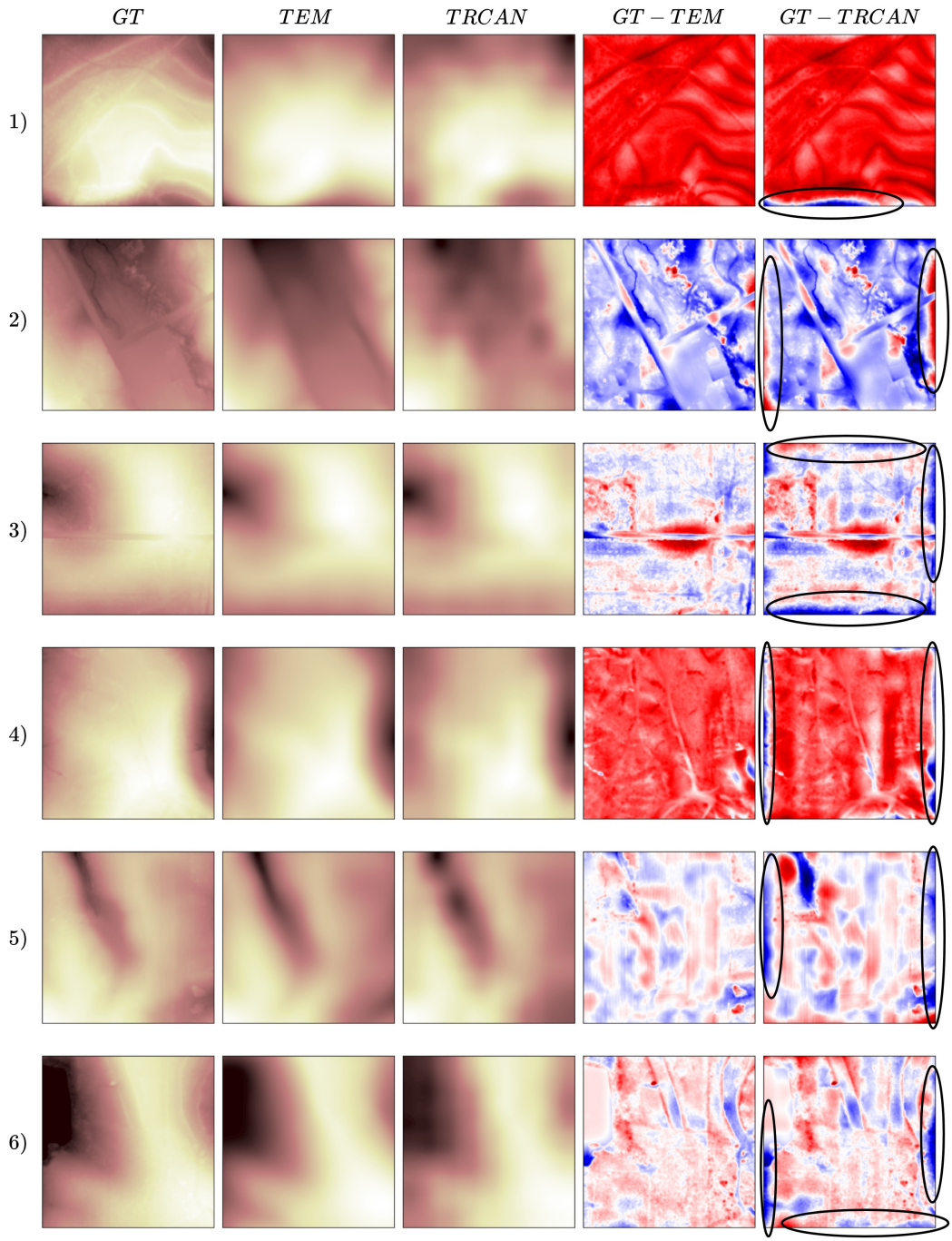


Figure 5.9: Illustration of error along the edge, comparing patch-wise method TRCAN with TEM for 6 randomly chosen samples.

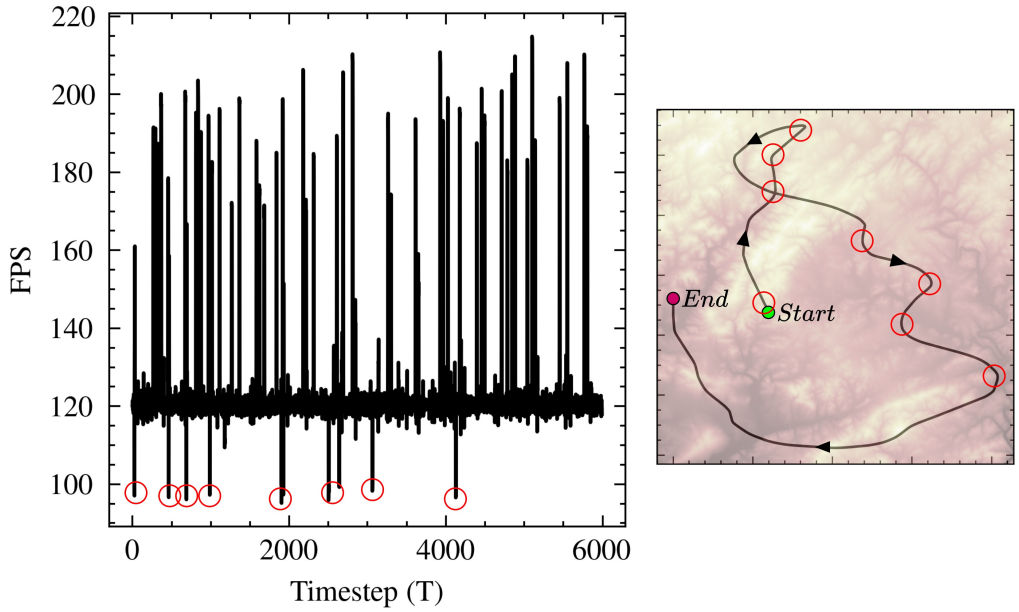


Figure 5.10: FPS (left) observed along a randomly traversed path (right). The red circles indicate the FPS dropping below 100.

of using an image pyramid in TEM effectively addresses the limitations of patch-based methods by incorporating global context.

5.3.8 Rendering Details

Our framework employs OpenGL and GLSL for rendering tasks in Python, enabling compatibility with PyTorch. In Figure 5.11, we showcase a real-time rendered frame that features terrain, trees, and water bodies. As illustrated in Figure 5.1a, the level of detail in the terrain is highlighted through a mesh representation. We utilize quad-trees for view frustum culling to achieve optimized performance, as demonstrated in Figure 5.1b in an overhead view of the frame. Additionally, we present a rendered scene generated and processed in our framework using Terragen [77], as shown in Figure 5.1c.

To evaluate the real-time performance of our model, we conducted experiments using a system with a NVIDIA GeForce RTX 3050 Laptop GPU and a 12th Gen Intel Core i5 CPU clocked at 2.50 GHz. In the absence of batching, we found that the TCM achieved 10 inferences in 0.32s on the GPU and 10 inferences in 3.44s on the CPU. Similarly, the TEM completed 100 inferences in 0.19s on the GPU and 1.70s on the CPU, all while maintaining real-time performance.

In addition, we profiled the system performance by traversing a random path and recording the frames per second (FPS) at different points. 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. However,

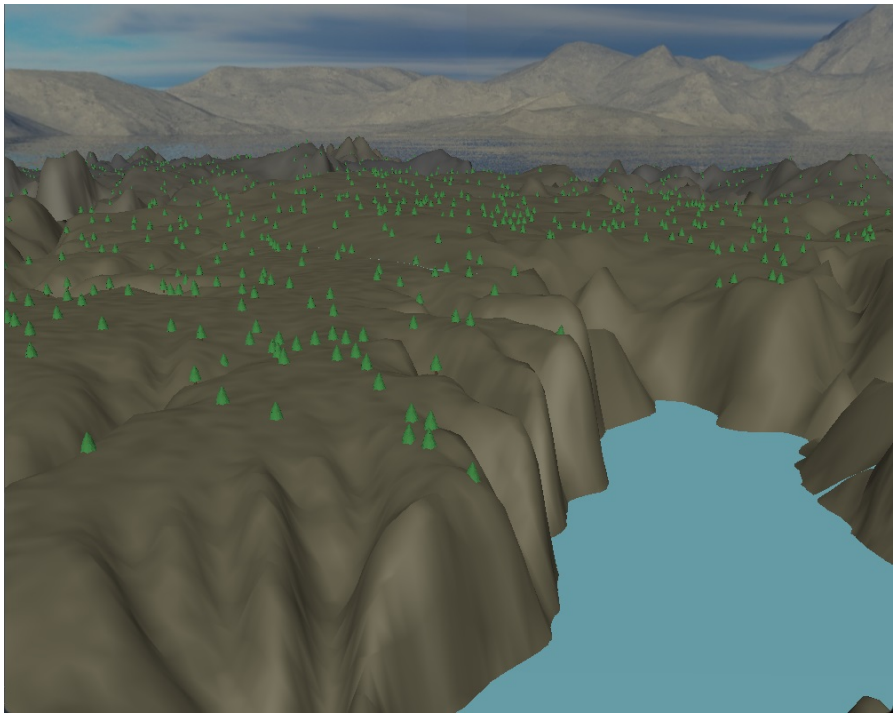


Figure 5.11: A view from our implementation of the proposed framework.

we confirm that these inferences did not exceed the real-time performance thresholds. These findings are summarized in Figure 5.10.

5.4 Conclusion and Future Work

In conclusion, our framework has advanced the state-of-the-art in terrain generation and rendering by providing a comprehensive solution that addresses the challenges of infinite terrain generation, efficient LODing, and seamless real-time 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 terrain rendering. Furthermore, the quadtree based enhancement module may be extended to other domains where the raster sizes are relatively larger, such as the medical domain. Another possible direction of exploration is the integration of other rendering algorithms such as ROAM or geographic mipmapping in a similar framework.

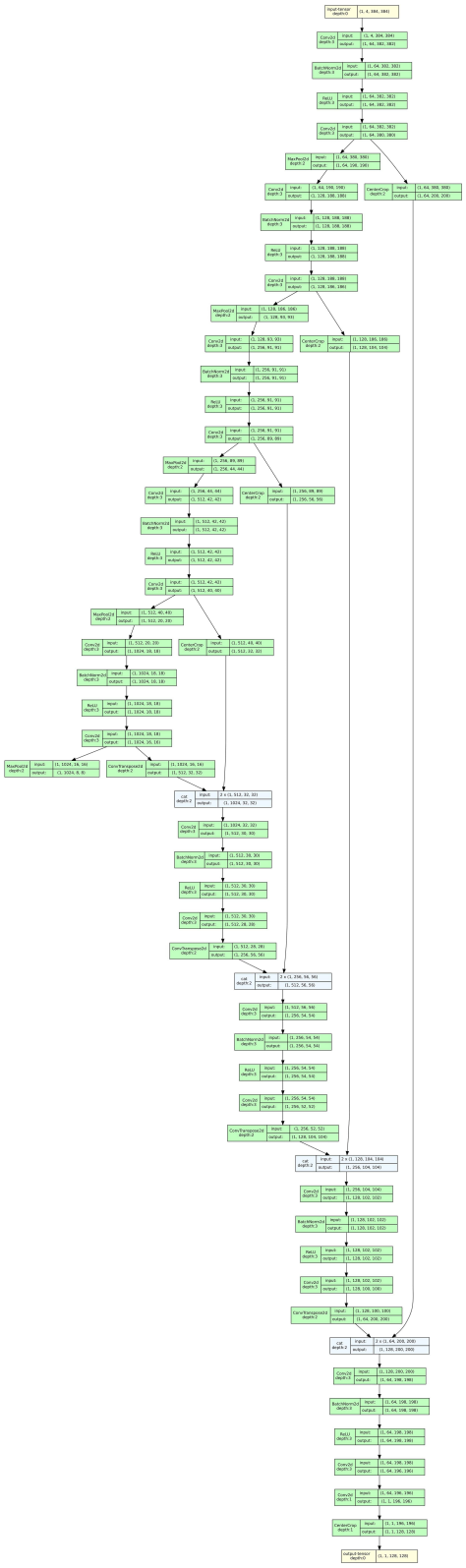


Figure 5.12: Architecture diagram of the generator (UNet) used in the TCM. Zoom in for details.

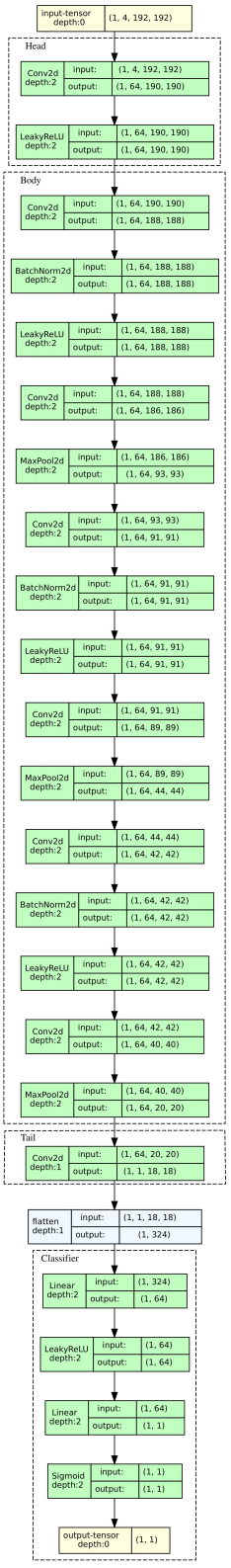


Figure 5.13: Architecture diagram of the discriminator used in the TCM. Zoom in for details.

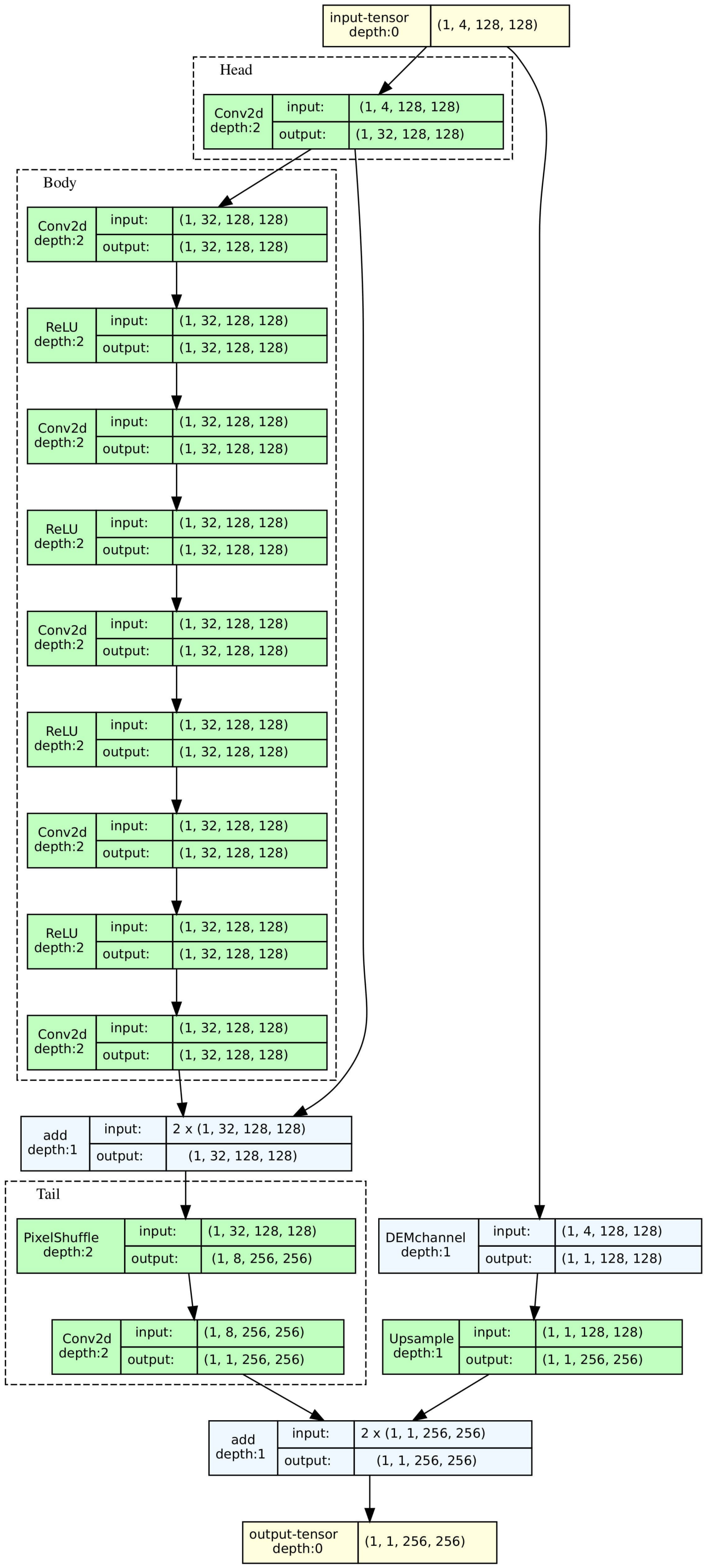


Figure 5.14: Architecture diagram of a single model in TEM.

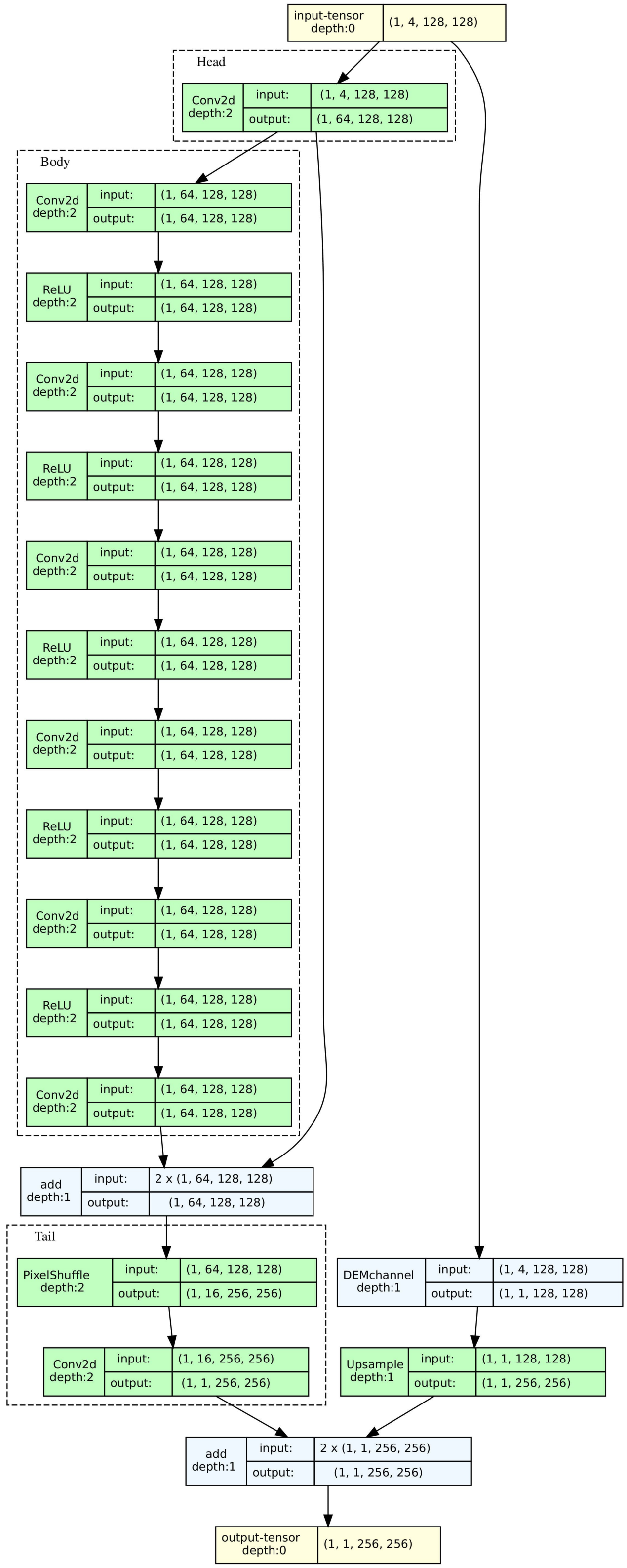


Figure 5.15: Architecture diagram of a single model in TEM_{big}.

Chapter 6

Conclusion and Future Work

In conclusion, this thesis has successfully discussed techniques to create fully realized virtual worlds with realistic terrain and trees. The motivation for this research was to create an algorithm that could produce a virtual world that could be used in gaming, simulation, and virtual reality applications.

We started the thesis by introducing the problem and providing motivation for our research. We also provided an overview of the necessary background topics required for understanding the subsequent chapters. The goal was to lay the groundwork for the research by identifying the problems that exist in current methods of generating virtual worlds and highlighting the need for more efficient and flexible algorithms.

We continued by discussing the generation and manipulation of terrain patches, which served as the building blocks for our virtual world. We described the various techniques and algorithms used for terrain generation. The method proved to be effective in creating visually appealing and realistic terrain. Furthermore, we extended the concept of terrain patches to generate infinite terrains using an adaptive and multi-resolution approach to ensure optimal performance. The multi-resolution approach provided a mechanism for adjusting the level of detail in different parts of the terrain, resulting in a more realistic and visually appealing virtual world.

This thesis also concentrated on the generation of trees, which enhanced the realism of the virtual world. We improved upon the interpretability and expressibility of L-systems in the process. We described the challenges associated with generating realistic trees and provided a solution that utilized L-systems to generate a variety of tree models.

Finally, we extended and improved upon the idea of generating infinite terrain and populating the terrain with trees. We laid down the steps involved in rendering a fully realized virtual world, which includes seamless integration with the terrain generation algorithm. We provide details of the rendering pipeline and how it was optimized to provide a smooth and visually pleasing user experience.

Although this thesis has made notable contributions to the field, there remain limitations that require attention in future research. One constraint of the current algorithm is its heavy reliance on hardware processing power, which may not be available in devices such as low-end mobiles or VR headsets. To

address this, future work could concentrate on developing more efficient algorithms that require fewer resources while still preserving the realism and complexity of the virtual world.

Furthermore, the virtual world generated by the algorithm is static and does not consider dynamic elements such as weather conditions, seasonal variations or erosion. To make the virtual world more immersive and responsive to changes in the environment, future research could incorporate dynamic elements. Additionally, this research focused primarily on generating terrain and trees, but future work could expand the algorithm to generate other natural features such as water bodies, clouds, and rocks, further enhancing the virtual world's realism. It's also worth noting that the absence of VR smell and touch technology can limit the user's sense of presence, despite the current algorithm's visually appealing and realistic terrain and trees. Future research could explore incorporating these sensory experiences to further enhance the user's immersion in the virtual environment.

In summary, this thesis shed light on algorithms that generate and renders a fully realized virtual world with realistic terrain and trees. This research contributes to and has significant applications in gaming, simulation, and virtual reality. Future work can build upon this research to further improve the realism and complexity of virtual worlds.

Related Publications

- *Automated tree generation using grammar & particle system*. Aryamaan Jain, Jyoti Sunkara, Ishaan Shah, Avinash Sharma, and K S Rajan. In Proceedings of the Twelfth Indian Conference on Computer Vision, Graphics and Image Processing, 2021.
- *Deep Generative Framework for Interactive 3D Terrain Authoring and Manipulation*. Shanthika Naik, Aryamaan Jain, Avinash Sharma, and K S Rajan. IEEE International Geoscience and Remote Sensing Symposium (IGARSS), 2022.
- *Adaptive & Multi-Resolution Procedural Infinite Terrain Generation with Diffusion Models and Perlin Noise*. Aryamaan Jain, Avinash Sharma, and K S Rajan. In Proceedings of the Thirteenth Indian Conference on Computer Vision, Graphics and Image Processing, 2022.
- *Real-Time Terrain Generation and Rendering*. Aryamaan Jain, Avinash Sharma, and K S Rajan. ACM International Conference on Multimedia, 2023. (*under review*)

Bibliography

[1] J. A. Alvarez-Cedillo, R. Almanza-Nieto, and J. C. Herrera-Lozada. Three dimensional hair model by means particles using blender. *3D Research*, 1(3):1–5, 2010.

[2] O. Argudo, C. Andujar, and A. Chica. Image-based tree variations. In *Computer Graphics Forum*, volume 39, pages 174–184. Wiley Online Library, 2020.

[3] O. Argudo, A. Chica, and C. Andujar. Terrain super-resolution through aerial imagery and fully convolutional networks. In *Computer Graphics Forum*, volume 37, pages 101–110. Wiley Online Library, 2018.

[4] O. Argudo, E. Galin, A. Peytavie, A. Paris, J. Gain, and E. Guérin. Orometry-based terrain analysis and synthesis. *ACM Transactions on Graphics (TOG)*, 38(6):1–12, 2019.

[5] Ó. Argudo Medrano. Realistic reconstruction and rendering of detailed 3d scenarios from multiple data sources. 2018.

[6] bisimulations. Vbs4. `https://bisimulations.com/products/vbs4`. Accessed: 2023-03-01.

[7] E. O. Brigham and R. Morrow. The fast fourier transform. *IEEE spectrum*, 4(12):63–70, 1967.

[8] N. Chiba, K. Muraoka, and K. 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.

[9] G. Cordonnier, E. Galin, J. Gain, B. Benes, E. Guérin, A. Peytavie, and M.-P. Cani. Authoring landscapes by combining ecosystem and terrain erosion simulation. *ACM Transactions on Graphics (TOG)*, 36(4):1–12, 2017.

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

[11] H. Designs. Central vista project, 2023.

[12] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 275–286, 1998.

[13] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

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

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

[16] A. Fournier, D. Fussell, and L. Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, 1982.

[17] E. Galin, E. Guérin, A. Peytavie, G. Cordonnier, M.-P. Cani, B. Benes, and J. Gain. A review of digital terrain modeling. In *Computer Graphics Forum*, volume 38, pages 553–577. Wiley Online Library, 2019.

[18] E. Galin, E. Guérin, A. Peytavie, G. Cordonnier, M.-P. Cani, B. Benes, and J. Gain. A review of digital terrain modeling. *Computer Graphics Forum*, 38(2):553–577, 2019.

[19] S. B. GeoKatalog. Tyrol.

[20] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[21] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[22] GSA. Spatial data management, 2023.

[23] É. Guérin, J. Digne, E. Galin, A. Peytavie, C. Wolf, B. Benes, and B. Martinez. Interactive example-based terrain authoring with conditional generative adversarial networks. *Acm Transactions on Graphics (TOG)*, 36(6):1–13, 2017.

[24] S. Gustavson. Simplex noise demystified. *Linköping University, Linköping, Sweden, Research Report*, 2005.

[25] E. Guérin, J. Digne, E. Galin, A. Peytavie, C. Wolf, B. Benes, and B. Martinez. Interactive example-based terrain authoring with conditional generative adversarial networks. *ACM Transactions on Graphics (ToG)*, 36(6), 2017.

[26] F. Han and S.-C. Zhu. Bayesian reconstruction of 3d shapes and scenes from a single image. In *First IEEE International Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis, 2003. HLK 2003.*, pages 12–20. IEEE, 2003.

[27] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[28] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[29] H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin. Feature based terrain generation using diffusion equation. In *Computer Graphics Forum*, volume 29, pages 2179–2186. Wiley Online Library, 2010.

[30] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[31] H. 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.

[32] I. C. i Geològic de Catalunya. Icgc – vissir3.

[33] ICC. Institut cartogràfic i geològic de catalunya (icc). `http://www.icc.cat/vissir3`. Accessed: February 2, 2019.

[34] S. Iizuka, E. Simo-Serra, and H. Ishikawa. Globally and locally consistent image completion. *ACM Transactions on Graphics (ToG)*, 36(4):1–14, 2017.

[35] P. Isola, J.-Y. Zhu, T. Zhou, and A. 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.

[36] A. Jain, A. 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.

[37] A. Jain, J. Sunkara, I. Shah, A. Sharma, and K. Rajan. Automated tree generation using grammar & particle system. In *Proceedings of the Twelfth Indian Conference on Computer Vision, Graphics and Image Processing*, pages 1–9, 2021.

[38] J. Kenwood, J. Gain, and P. Marais. Efficient procedural generation of forests. 2014.

[39] Khronos. Glsl, 2023.

[40] J. Kim. Modeling and optimization of a tree based on virtual reality for immersive virtual landscape generation. *Symmetry*, 8(9):93, 2016.

[41] L. King. Lion king, 2023.

[42] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[43] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[44] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2014.

[45] J. Klein, S. Hartmann, M. Weinmann, and D. L. Michels. Multi-scale terrain texturing using generative adversarial networks. In *2017 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pages 1–6, 2017.

[46] A. Kubade, D. Patel, A. Sharma, and K. Rajan. Afn: Attentional feedback network based 3d terrain super-resolution. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[47] A. A. Kubade, A. Sharma, and K. 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.

[48] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. 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.

[49] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. 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.

[50] LearnOpenGL. Compute shaders, 2023.

[51] LearnOpenGL. Geometry shaders, 2023.

[52] LearnOpenGL. Lighting, 2023.

[53] LearnOpenGL. Shaders, 2023.

[54] LearnOpenGL. Tesselation shaders, 2023.

[55] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. 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.

[56] H. Li, X. Tuo, Y. Liu, and X. 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.

[57] A. Lindenmayer. Mathematical models for cellular interactions in development ii. simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology*, 18(3):300–315, 1968.

[58] E. Linder-Norén. GitHub - eriklindernoren/PyTorch-GAN: PyTorch implementations of Generative Adversarial Networks. — github.com. `https://github.com/eriklindernoren/PyTorch-GAN`, 2021. [Accessed 02-Sep-2022].

[59] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. 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.

[60] S. Longay, A. Runions, F. Boudon, and P. Prusinkiewicz. Treesketch: Interactive procedural modeling of trees on a tablet. In *SBIM@ Expressive*, pages 107–120. Citeseer, 2012.

[61] F. Losasso and H. Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. In *ACM Siggraph 2004 Papers*, pages 769–776. 2004.

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

[63] Microsoft. Microsoft flight simulator, 2023.

[64] Minecraft. Minecraft, 2023.

[65] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[66] V. Mulder, S. De Bruin, M. E. Schaepman, and T. Mayr. The use of remote sensing in soil and terrain mapping—a review. *Geoderma*, 162(1-2):1–19, 2011.

[67] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. *ACM Siggraph Computer Graphics*, 23(3):41–50, 1989.

[68] S. Naik, A. Jain, A. 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.

[69] A. Q. Nichol and P. Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

[70] R. Pajarola. Overview of quadtree-based terrain triangulation and visualization. 2002.

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

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

[73] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

[74] Pbr-book.org, 2022.

[75] K. Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.

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

[77] Planetside. Terragen 4, 2023.

[78] L. Quan, P. Tan, G. Zeng, L. Yuan, J. Wang, and S. B. Kang. Image-based plant modeling. In *ACM SIGGRAPH 2006 Papers*, pages 599–604. 2006.

[79] W. T. Reeves. Particle systems—a technique for modeling a class of fuzzy objects. *ACM Transactions On Graphics (TOG)*, 2(2):91–108, 1983.

[80] D. Ritchie, B. Mildenhall, N. D. Goodman, and P. Hanrahan. Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. *ACM Transactions on Graphics (TOG)*, 34(4):1–11, 2015.

[81] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015:, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[82] T. J. Rose and A. G. Bakaoukas. Algorithms and approaches for procedural terrain generation-a brief review of current techniques. In *2016 8th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, pages 1–2. IEEE, 2016.

[83] S. Röttger, W. Heidrich, P. Slusallek, and H.-P. Seidel. Real-time generation of continuous levels of detail for height fields. 1998.

[84] T. Sakaguchi. Botanical tree structure modeling based on real image set. In *ACM SIGGRAPH 98 Conference abstracts and applications*, page 272, 1998.

[85] M. Santini, S. Grimaldi, F. Nardi, A. Petroselli, and M. C. Rulli. Pre-processing algorithms and landslide modelling on remotely sensed dems. *Geomorphology*, 113(1-2):110–125, 2009.

[86] SBG. Südtiroler bürgernetz geokatalog (sbg). `http://geokatalog.buergernetz.bz.it/geokatalog`. Accessed: February 2, 2019.

[87] E. Scrolls. Elder scrolls, 2023.

[88] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. 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.

[89] I. Shlyakhter, M. Rozenoer, J. Dorsey, and S. Teller. Reconstructing 3d tree models from instrumented photographs. *IEEE Computer Graphics and Applications*, 21(3):53–61, 2001.

[90] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[91] O. Šťava, B. Beneš, R. Měch, D. G. Aliaga, and P. Krištof. Inverse procedural modeling by automatic generation of l-systems. In *Computer Graphics Forum*, volume 29, pages 665–674. Wiley Online Library, 2010.

[92] O. Stava, S. Pirk, J. Kratt, B. Chen, R. Mvech, O. Deussen, and B. Benes. Inverse procedural modelling of trees. In *Computer Graphics Forum*, volume 33, pages 118–131. Wiley, 2014.

[93] R. Sun, J. Jia, and M. Jaeger. Intelligent tree modeling based on l-system. In *2009 IEEE 10th International Conference on Computer-Aided Industrial Design & Conceptual Design*, pages 1096–1100. IEEE, 2009.

[94] T. M. Takala, M. Mäkäräinen, and P. Hämäläinen. Immersive 3d modeling with blender and off-the-shelf hardware. In *2013 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 191–192. IEEE, 2013.

[95] P. Tan, G. Zeng, J. Wang, S. B. Kang, and L. Quan. Image-based tree modeling. In *ACM SIGGRAPH 2007 papers*, pages 87–es. 2007.

[96] USGS. 1 meter digital elevation models (dems) - usgs national map 3dep downloadable data collection. `https://www.sciencebase.gov/catalog/item/543e6b86e4b0fd76af69cf4c`. Accessed: 2023-01-01.

[97] Y. Wang, F. Perazzi, B. McWilliams, A. Sorkine-Hornung, O. Sorkine-Hornung, and C. 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.

[98] J. Weber and J. Penn. Creation and rendering of realistic trees. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119–128, 1995.

[99] Wikipedia. Dem, 2023.

[100] Wikipedia. Gis, 2023.

[101] Wikipedia. Graphics pipeline, 2023.

[102] Wikipedia. Lidar, 2023.

[103] Wikipedia. Pathtracing, 2023.

[104] Wikipedia. Rasterization, 2023.

[105] Wikipedia. Raytracing, 2023.

[106] C. Wojtan, M. Carlson, P. J. Mucha, and G. Turk. Animating corrosion and erosion. In *NPH*, pages 15–22, 2007.

[107] J. Wu, Y. Yang, S.-r. Gong, and Z. Cui. A new quadtree-based terrain lod algorithm. *J. Softw.*, 5(7):769–776, 2010.

[108] H. Xiang, Q. Zou, M. A. Nawaz, X. Huang, F. Zhang, and H. Yu. Deep learning for image inpainting: A survey. *Pattern Recognition*, 134:109046, 2023.

[109] Q. Xiao, G. Li, and Q. Chen. Image outpainting: Hallucinating beyond the image. *IEEE Access*, 8:173576–173583, 2020.

[110] K. Xie, F. Yan, A. Sharf, O. Deussen, H. Huang, and B. Chen. Tree modeling with real tree-parts examples. *IEEE transactions on visualization and computer graphics*, 22(12):2608–2618, 2015.

[111] M. E. Yumer, P. Asente, R. Mech, and L. B. Kara. Procedural modeling using autoencoder networks. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 109–118, 2015.

[112] L. Zhang, J. She, J. Tan, B. Wang, and Y. Sun. A multilevel terrain rendering method based on dynamic stitching strips. *ISPRS International Journal of Geo-Information*, 8(6):255, 2019.

[113] L. Zhang, P. Wang, C. Huang, B. Ai, and W. Feng. A method of optimizing terrain rendering using digital terrain analysis. *ISPRS International Journal of Geo-Information*, 10(10):666, 2021.

[114] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu. Image super-resolution using very deep residual channel attention networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 286–301, 2018.

[115] 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.

[116] Y. Zhao, H. Liu, I. Borovikov, A. Beirami, M. Sanjabi, and K. Zaman. Multi-theme generative adversarial terrain amplification. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019.

[117] H. Zhou, J. Sun, G. Turk, and J. M. Rehg. Terrain synthesis from digital elevation models. *IEEE transactions on visualization and computer graphics*, 13(4):834–848, 2007.