Contents

Abstract

This thesis, entitled *"Procedural terrain generation for underwater environments"*, as its name suggests, focuses on the topic of procedural terrain generation with the specificity to tackle the tackle underwater environments. Terrain generation is still an open research area in computer graphics as, with the emergence of simulation, rendering, and interaction techniques, the entire field is experiencing increased collaboration with terrain experts. Virtualization of the physical world helps users to see and manipulate it, allowing for a better understanding of it and bringing together numerous experts, gradually breaking down the boundaries of scientific disciplines. Terrain science brings together, in almost a direct manner, geologists, oceanologists, physicists, meteorologists, biologists, roboticists, computer scientists, 3D artists, and more. We focused our work on the inclusion of the user in the generation process of virtual worlds through fast and controlable algorithms, with the possibility to dive underwater.

This thesis is divided into three parts, guiding the user from the design of a landscape, its 3D modeling, until its finalization, step by step. Firstly, we will propose a formalization of terrain designing, allowing for the conception of environments in a semantic sense, abstracting away from 3D geometry and data structures.

Secondly, we will see how to give 3D form to these environments. We will propose new models for the generation and modeling of coral islands and karst networks.

In the third part, we will focus on adding realism to 3D terrains through physical simulations of erosion processes. We will demonstrate a flexible and controllable method to imitate the long-term effects of water and wind on both terrestrial and marine landscapes.

Keywords: terrain representation, procedural generation, physical simulations, user interaction

CHAPTER 1

Introduction

Over the past 50 years, terrain generation has emerged as an increasingly active domain inside the field of computer graphics. As the demand for more realistic, faster, and automated processes has grown, terrain generation techniques have evolved to meet these challenges. The focus of this work is on one specific branch of terrain generation: the interactive creation and modeling of landscapes.

[TODO]

What sets this work apart is its emphasis on underwater landscapes or "seascapes", an area still overviewed but with an important relevance in marine biology, oceanography, and underwater robotics. Furthermore, the extension of these techniques to new areas of the entertainment industry, such as video games and cinema, presents new opportunities for innovation.

[TODO]

The central question guiding this research is: "How can we efficiently guide the user in the creation of virtual content along the production process line to maintain as much control as possible over the final product?" This concept of "guiding" rather than "replacing" the user is, from my point of view, fundamental, as no machine can truly know better than a human what the final product should look like. The goal is to present algorithms that can be used flexibly within a production pipeline, adapting to many terrain representation, fluid solver, landscape type, hardware, or objective, whether the final use is real-time rendering, realism, or animation.

Maintaining control over the creative process is essential. Each generated result should feel unique, meet the user's expectations, be explainable, and be

easily correctible without requiring a complete regeneration. This approach ensures that the user remains at the center of the creative process, with the tools and algorithms serving to enhance their objectives, rather than replace them.

1.1 Contributions and outlines

This research follows a chronological order of terrain generation, beginning from the development of an abstract representation that bridges the gap between computer science and Earth science. This approach offers a generalization of the desired landscape type, with a particular focus on underwater environments, but also applicable to terrestrial landscapes.

A second key contribution is the proposal of new types of landscape features, including karst networks and coral islands, which maintain the notion of sparseness through the use of implicit volumes.

Finally, a new particle-based erosion simulation method is introduced. This method is agnostic to the terrain representation, making it lightweight, fast, and easy to implement.

The overarching aim is to provide maximum control to the user, not only during the generation process but also in making corrections to details upstream, ensuring that the final product aligns with the user's vision.

1.1.1 Semantics

[TODO] This interdisciplinary approach enriches the modeling process, ensuring that the generated landscapes reflect real-world biological and geological phenomena.

1.1.2 Modeling

[NOT GOOD AT ALL, TO REDO]

The generation of certain landscape elements, such as karst networks and coral islands, remains relatively new. While karsts have been explored in previous research (Axel Paris), the representation of coral islands, particularly in the context of Darwin's theories, is an innovative contribution. Karst networks are represented in a highly user-friendly manner, using a directed acyclic graph structure, which closely resembles a tree structure. This method is enhanced by a fractal generation approach with cycles, allowing for iterative generation of complex landscapes. The modeling of coral islands is based on both historical observations and travel journals.

1.1.3 Amplification

To increase the realism of the generated landscapes, details are added through a particle-based erosion method. This method is designed to be generalizable for flexibility, and its implementation is oriented towards speed and parallelization.

These contributions collectively aim to advance the field of terrain generation, providing new tools and methods that offer both flexibility and control to users, while also pushing the boundaries of what is possible in the simulation and modeling of natural landscapes.

1.2 Terrain representations

Terrain refers to the physical features and configuration of a specific area of land. It includes the elevation, slope, and the overall topography, such as mountains, valleys, and plains. Terrain is often used to describe the surface characteristics of the land, focusing on the natural contours and the geographical aspects that define a region's physical form.

While the term "terrain" describes the physical characteristics of land, it does not include the natural elements that shape an area's identity. Elements like vegetation, water bodies, and climatic conditions, such as snow cover, are essential to how we perceive and understand a landscape. Therefore, when discussing procedural generation in virtual environments, "landscape generation" is a more fitting term, as it integrates these natural elements along with the topographical features.

In addition to "terrain generation," other terms such as "landscape generation," "world generation," and "environment generation" can be used to describe the creation of virtual landscapes. These terms are interchangeable and can all refer to the process of generating physical terrain along with natural and artificial elements. However, by convention and for simplicity, the term "terrain generation" is most commonly used in the field. Despite its original focus on the physical features of the land, "terrain generation" has evolved to encompass a broader range of environmental elements, making it a convenient and widely accepted term for describing the comprehensive process of creating virtual environments.

A terrain can be represented in various ways, each of them suited for a given application of which we give an brief overview, more details can be found in (E. Galin et al., 2019).

1.2.1 Elevation models

Elevation models are a fundamental approach in terrain representation, widely used in procedural generation due to their simplicity and efficiency. These models define the terrain as a function $h : \mathbb{R}^2 \mapsto \mathbb{R}$, where each point in a 2D plane is mapped to an elevation value. This approach is particularly effective for representing terrains where the elevation is the only varying factor, such as hills, valleys, and plateaus, and it is best suited for terrains without complex 3D features like overhangs or caves. While we visualize elevation models in three dimensions, they are mathematically considered two-dimensional functions. In the domain of terrain generation, we will name them 2.5D models.

Elevation models are widely used in industries where large-scale terrain representation is crucial. In video games, they provide the foundation for creating vast open-world environments. In geographic information systems (GIS) and remote sensing, height fields are used to represent real-world terrain data, offering a practical means of visualizing and analyzing geographical features. The ability to manipulate and control terrain features procedurally makes elevation models a common choice for applications that require efficient terrain generation and rendering.

They offer a powerful method for representing terrains in procedural generation, combining simplicity with flexibility. While they have limitations in representing complex 3D structures, their efficiency and compatibility with existing algorithms make them indispensable in a variety of applications.

Implicit height fields

Implicit height fields represent the terrain as a mathematical function that provides a height value at any given point in the domain. These functions can be procedural or closed-form expressions, allowing for compact storage and infinite precision in theory. The elevation function allows for easy manipulation of terrain features, making it ideal for generating terrains that require smooth, continuous surfaces. However, the primary disadvantage is the computational complexity involved in evaluating the function, especially for large or highly detailed terrains. The challenge lies in constructing functions that can realistically represent large-scale terrains with complex landforms.

Discrete height fields

Discrete height fields, or explicit height fields, are one of the most prevalent methods for terrain representation. These models consist of a 2D grid where each cell contains a height value, representing the elevation at that point. Height fields

are particularly advantageous because they are simple to implement and are directly compatible with many rendering techniques and hardware, but also due to their closeness with image processing, a domain studied for many decades now.

The main advantage of height fields is their ability to handle large datasets efficiently, providing a balance between memory usage and detail. However, they are limited by their inability to represent terrains with overhangs or caves, as each point on the grid can only hold a single elevation value. Additionally, height fields often require interpolation methods, such as bi-linear or bi-cubic interpolation, to reconstruct a continuous surface from the discrete grid points.

1.2.2 Volumetric models

Volumetric models represent a more complex approach to terrain modeling, allowing for the depiction of 3D features that go beyond the simple surface-based representation provided by elevation models. These models capture not only the surface of the terrain but also its internal structure, making them ideal for representing terrains with overhangs, caves, and other subsurface features.

Volumetric models, including layered materials, voxel grids, and implicit models, are essential in applications where terrain complexity and detail are primordial. In geological simulations, these models allow for accurate representation of subsurface structures and processes. Voxel models are widely used in games that require dynamic terrain deformation, providing a rich interactive environment for players. Implicit models are favored in situations where smooth, continuous surfaces are needed [FIND OTHER USE CASES].

Implicit volumetric models

Implicit volumetric models describe the terrain's shape and features using an implicit function. The terrain is represented by a mathematical function $f : \mathbb{R}^3 \mapsto \mathbb{R}$ that determines the terrain surface by evaluating to an isovalue, often zero. This function provides a continuous representation of the terrain, with points inside the terrain returning positive values and while points in the air evaluate to negative values. It allows for the seamless representation of complex terrain features, including caves, overhangs, and varying geological structures, which are impossible to represent with elevation models.

One of the key advantages of implicit models is their ability to produce smooth surfaces without the need for discrete polygonal meshes, which can result in realistic and natural-looking terrains. However, the computational complexity of evaluating the implicit function, especially for large terrains, can

be a significant drawback. Additionally, converting an implicit surface into a mesh for rendering can be challenging and resource-intensive.

Layered models

Layered models are a type of volumetric representation that encode different material layers within the terrain and are defined by a function $\mu : \mathbb{R}^3 \mapsto M$, where M denotes the material type at any given point in 3D space. This allows for a detailed representation of the terrain's internal composition, which can be crucial for applications requiring realistic geological simulations. Each layer is defined by its thickness or elevation, and multiple layers can be stacked to represent complex geological formations. These layers might include materials like bedrock, sand, soil, or water, each contributing to the overall structure of the terrain. Layered models are particularly useful in simulations that involve processes like erosion or sedimentation, where the interaction between different material layers affects the physical process.

The primary advantage of layered models is their ability to represent a stratified terrain with distinct material properties, which can be manipulated individually. This makes them well-suited for simulations that require detailed geological accuracy. However, they are more complex to implement than simple elevation models and require additional computational resources to manage the interactions between layers.

Voxel grid models

Voxel grids are a common method for representing 3D terrains in procedural generation, offering the ability to capture complex internal structures and features that are difficult or impossible to represent with surface-based models. In a voxel grid, the 3D space is divided into a regular grid of small, cube-shaped elements called voxels (volumetric pixels). Each voxel holds information about the material or properties of the terrain at that specific point in space. This approach allows for detailed modeling of features such as caves, tunnels, overhangs, and intricate underground networks. The regular grid structure allows for the use of image processing-oriented algorithms.

There are three primary types of voxel grids used in terrain representation: binary voxel grids, material voxel grids and density voxel grids. Each has distinct characteristics, advantages, and limitations, making them suitable for different applications.

Binary voxel grids

Binary voxel grids are the simplest form of voxel representation. In these grids, defined $f : \mathbb{R}^3 \mapsto [0, 1]$, each voxel is either "filled" or "empty," representing the presence or absence of material. This binary state is typically represented by a 1 (filled) or 0 (empty). Binary voxel grids are straightforward to implement and require much less memory compared to more complex voxel representations, making them ideal for applications where the primary concern is whether a space is occupied or not.

The simplicity of binary voxel grids is one of their main advantages. They are easy to understand and visualize, with each voxel requiring only a single bit of information to represent its state. Additionally, because only a binary state is stored, these grids can be memory-efficient when combined with compression techniques like Sparse Voxel Octrees (SVOs) or voxel Directed Acyclic Graphs (DAG). The simplicity of the data structure also allows for quick processing, making binary voxel grids suitable for real-time applications where performance is required. However, the binary nature of these grids limits their ability to represent variations in material density or properties, or even smoothness, resulting in less detailed terrain models. This can lead to hard, blocky edges in the terrain, which may appear unnatural without additional smoothing or processing.

Material voxel grids

Material voxel grids, defined as $\mu : \mathbb{R}^3 \mapsto M$, are commonly used in applications where simple occupancy information is sufficient. For example, voxel-based games like Minecraft utilize material grids to create terrains composed of solid blocks with clear boundaries. These grids are also employed in scientific simulations where the primary concern is the presence or absence of materials, rather than detailed material properties.

Density voxel grids

Finally, density voxel grids allow each voxel to store a range of values, representing varying degrees of material presence with $f : \mathbb{R}^3 \mapsto \mathbb{R}$. Instead of a simple discrete state, a density voxel grid assigns a continuous value to each voxel, which can represent material density, opacity, or other properties. This added complexity enables density voxel grids to represent subtle variations in terrain, such as gradual changes in material density or smooth transitions between solid and empty spaces, allowing for more realistic and natural-looking terrain models.

Density voxel grids are often used in high-fidelity simulations where detail and realism are essential. They are found in applications such as medical imaging, scientific visualizations, and advanced terrain modeling for films and visual effects. These grids are also employed in procedural terrain generation systems that require smooth and natural transitions between different terrain features,

such as caves, cliffs, and eroded landscapes.

1.3 Procedural terrain generation

Procedural generation is a computational method used in computer science to create data algorithmically rather than manually, enabling the automatic generation of large amounts of content with minimal human input. This approach is crucial in fields such as game development, where it helps create large, varied worlds, and in simulations or data generation where diverse scenarios or datasets are needed. The process typically involves defining rules and algorithms that dictate how content is generated, ensuring it meets specific criteria and patterns, often incorporating randomness through noise functions to produce unique results each time. Incorporating adjustable parameters and customizable rules in the algorithms allows users to influence the characteristics and outcomes of the generated content. This method enhances efficiency, creativity, and scalability in digital content creation. However, one of the main challenges is ensuring that content generated by a single algorithm remains both diverse and coherent, but the main challenges are ensuring the generated content from a single algorithm is both diverse and coherent and achieving a balance between speed, realism and control to satisfy the desired design and quality.

1.3.1 Definition

Procedural generation is a powerful technique for creating data algorithmically, rather than manually. This method is massively used in areas such as computer graphics, simulations, and game development. Essentially, it involves using predefined rules or algorithms to generate complex structures or systems. For example, it can be employed to create landscapes, textures, or even entire worlds.

One of the main benefits of procedural generation is its data independence. Content is typically generated in real-time or on-the-fly, rather than being explicitly stored, which allows for the creation of large and dynamic environments without requiring significant storage space. However, procedural generation is also advantageous for producing diverse content quickly, enabling the creation of vast amounts of data that may require only minor adjustments or improvements.

In practical applications, procedural generation is commonly used in video games. It enables the creation of vast, varied, and detailed environments efficiently, without requiring large storage. For instance, games like *Minecraft* use procedural generation to create expansive worlds with diverse biomes, while titles like *No Man's Sky* generate entire galaxies filled with unique planets. This

automated approach involves defining a set of rules or procedures that produce diverse outputs, making the content creation process both dynamic and flexible. The generated content can adapt and change in response to different inputs or conditions, which is particularly valuable for applications needing variability and adaptability.

Beyond terrain generation, procedural generation is widely employed in various aspects of digital content creation across multiple industries. In video games, it is used for character animation, allowing for the automatic generation of diverse character movements and behaviors that respond dynamically to the environment. Level design benefits from procedural techniques by creating vast, intricate game levels that are unique with each playthrough, enhancing replayability. Texturing in games also utilizes procedural generation to produce varied surface details, such as the appearance of weathered stone or realistic skin textures, for example.

In the film industry, procedural generation plays a crucial role in generating realistic terrains for expansive outdoor scenes that would be time-consuming and costly to model by hand. It is also used in facial animation, where algorithms can create subtle, lifelike expressions that enhance the realism of CGI characters. Crowd simulations are another widely used application, where procedural techniques generate large numbers of unique characters with varied appearances and behaviors, populating scenes with realistic and dynamic crowds.

In simulations, procedural generation is integral to creating accurate representations of real-world environments. This includes generating complex ecosystems, urban layouts, and geological formations for scientific research, training simulations, and virtual reality applications. Additionally, procedural methods are used to isolate and control environmental variables, allowing researchers to test different scenarios and observe outcomes in a controlled, repeatable manner.

The integration of algorithms and data is a key aspect of procedural generation. It combines mathematical models, noise functions, and other algorithms to produce both realistic and abstract content. This might involve generating natural features like landscapes or textures, or even entire ecosystems. Incorporating elements of real-world phenomena, such as erosion patterns in terrain generation, can make the environments more believable and interactive.

Moreover, procedural generation allows for user-driven customization. Users can influence or guide the content generation process, leading to user-specific outcomes. This feature, combined with the scalability and efficiency of procedural methods, means that large amounts of data can be produced with relatively low computational and storage costs compared to manually crafted content.

Procedural generation can be categorized into two main types: deterministic

and stochastic systems.

Deterministic systems produce outputs that are entirely predictable and repeatable given the same initial parameters and input conditions. This means that running the same algorithm with identical inputs multiple times generates the exact same output. This predictability is valuable in scenarios where consistency and reliability are essential, such as in engineering simulations or when reproducing specific results for debugging purposes in game development. For example, in a video game, a deterministic procedural system might be used to generate the same game world across different sessions, ensuring that all players experience the same environment under the same conditions. Or in content creation pipelines, where artists or designers might want to fine-tune specific aspects of the generated content. Since the output is repeatable, they can iteratively adjust parameters and see how these changes affect the final result, making it easier to achieve the desired outcome.

Stochastic systems, on the other hand, introduce elements of randomness into the generation process, resulting in varied outputs even when the same initial parameters are used. This randomness allows for the creation of diverse and unpredictable content, which is particularly important in applications where variation and uniqueness are desired. For instance, in procedural content generation for video games, stochastic systems might be employed to generate a wide variety of levels, landscapes, or in-game items, ensuring that each playthrough feels different.

In many cases, procedural generation systems may combine deterministic and stochastic elements to leverage the strengths of both approaches. For example, a game might use deterministic algorithms to generate the overall structure of a level, ensuring certain key elements are always present, while using stochastic processes to populate the level with varied details and features, less critical in the application. This hybrid approach can provide a balance between consistency and variability, making it possible to create dynamic content that remains coherent.

Rule-based systems are another aspect of procedural generation, using predefined rules to generate content. This approach allows for controlled and structured outputs. Typically, procedural generation involves iterative processes, where initial results are refined or adjusted based on additional rules or parameters.

The advantages of procedural generation include efficiency, as it reduces the need for significant manual efforts, and variability, as it can produce a wide range of unique outputs from the same set of rules. Additionally, it is adaptable, easily responding to changes in requirements or user input, and it minimizes storage needs by generating content on-the-fly.

However, there are challenges associated with procedural generation. The complexity of developing and fine-tuning algorithms can be significant, requiring careful balancing of parameters. Striking a balance between realistic content and computational performance can also be difficult. Furthermore, meeting user expectations for content quality and variety, especially in interactive applications, can be a challenge.

1.3.2 History

The history of procedural generation can be traced back to the mid-20th century, rooted in mathematical and algorithmic theories. During this period, foundational concepts such as randomness, fractal geometry, and noise functions were introduced, laying the groundwork for modern procedural techniques.

Noises

One of the earliest and most significant developments in procedural generation was the introduction of noise functions. Noise functions are mathematical constructs used to generate pseudo-random, yet smooth and coherent, patterns. These patterns are critical in creating natural-looking textures and landscapes in computer graphics.

Ken Perlin introduced Perlin noise in 1983, initially for the CGI movie *Tron* (1982), a groundbreaking method for generating smooth, gradient-based noise. Perlin noise can be defined as a continuous function $P(\mathbf{p})$ that takes a point $\mathbf{p}$ in n-dimensional space and returns a scalar value that represents the noise intensity at that point. The function is defined as a sum of several layers of noise, called "octaves," each with its own frequency and amplitude:

$$P(\mathbf{p}) = \sum_{i=0}^{n-1} a_i \cdot \text{noise}(f_i \cdot \mathbf{p}) \tag{1.1}$$

where n is the number of octaves, a_i is the amplitude of the i-th octave, f_i is the frequency of the i-th octave.

The primary application of Perlin noise was in generating textures that mimic natural phenomena, such as clouds, fire, and marble. Perlin noise's key advantage was its ability to produce visually appealing, continuous gradients that avoided the harsh randomness seen in earlier noise functions. This made it an essential tool in computer graphics, particularly in texture mapping and terrain generation.

Building on his earlier work, Ken Perlin developed Simplex noise in 2001, which was designed to address some of the limitations of Perlin noise. Simplex noise is a more computationally efficient and visually isotropic alternative. It operates on a triangular (in 2D) or tetrahedral (in 3D) grid, reducing computational complexity, particularly in higher dimensions. The Simplex noise function, $S(\mathbf{p})$, can be defined similarly to Perlin noise but uses a different gradient selection and interpolation process that results in fewer directional artifacts:

$$S(\mathbf{p}) = \sum_{i=1}^{n} a_i \cdot \text{simplex\_noise}(f_i \cdot \mathbf{p}) \tag{1.2}$$

Simplex noise's reduced computational cost and smoother, isotropic results made it popular in real-time graphics applications, such as video games.

Wavelet noise uses wavelet transforms to control the frequency content of the noise across different scales, reducing artifacts like aliasing and improving texture fidelity. It is expressed as:

$$\text{Wavelet\_Noise}(\mathbf{p}) = \sum_{i=0}^{n-1} \text{Wavelet\_Transform}(a_i \cdot \text{noise}(f_i \cdot \mathbf{p})), \tag{1.3}$$

allowing for detailed and consistent textures at multiple resolutions.

Worley noise, introduced by Steven Worley, generates patterns based on the distances to randomly distributed points in space. This noise is particularly effective for creating cellular patterns, useful in simulating materials like stone or skin. The noise at a point $\mathbf{p}$ is computed as:

$$W(\mathbf{p}) = \min_i \{\text{distance}(\mathbf{p}, \mathbf{q}_i)\}, \tag{1.4}$$

where $\mathbf{q}_i$ are the feature points.

Curl noise is particularly useful for simulating fluid-like motion, such as swirling patterns in water or smoke. It is generated by taking the curl of a vector field derived from a noise function, resulting in a divergence-free flow. The curl noise is mathematically described as:

$$\mathbf{C}(\mathbf{p}) = \nabla \times \mathbf{F}(\mathbf{p}), \tag{1.5}$$

where $\mathbf{F}(\mathbf{p})$ is a vector field generated by the noise function.

Subdivision

In procedural generation, fractal geometry can be used to create natural-looking terrains and landscapes by generating structures that appear consistent across various scales. For example, the Diamond-Square algorithm, a fractal-based terrain generation technique, iteratively refines a coarse grid of points by introducing random variations. This process creates self-similar structures that resemble mountainous terrains:

$$h(x,y) = \frac{h(x_1,y_1) + h(x_2,y_2) + h(x_3,y_3) + h(x_4,y_4)}{4} + \text{random\_offset} \quad (1.6)$$

where $h(x,y)$ is the height at a given point, and $(x_1,y_1), (x_2,y_2), (x_3,y_3), (x_4,y_4)$ are the surrounding points.

Fractal-based methods revolutionized terrain generation by providing a mathematical framework for creating realistic and complex natural landscapes.

Entertaining industry

As procedural generation techniques matured, they found early applications in video games and interactive media. One of the first notable examples was the game *Rogue* (1980), which utilized procedural generation to create its dungeon levels. In Rogue, each dungeon was generated anew each time a player started a game, ensuring that no two playthroughs were alike. This approach not only enhanced replayability but also minimized the amount of storage needed, as the game did not store pre-made levels but rather generated them on-the-fly.

Another milestone in procedural generation was the game *Elite* (1984), which utilized procedural algorithms to generate a massive universe containing 2048 unique planets, each with its own name, economy, and location in space. This vast universe was generated using only about 20KB of memory, thanks to the efficiency of the procedural techniques employed. The planets' attributes were generated using deterministic algorithms, ensuring that each planet's characteristics were consistent across different game sessions.

Dwarf Fortress (2006) took procedural generation to new heights by generating not just terrain, but entire worlds, complete with histories, civilizations, and ecosystems. The game uses complex algorithms to simulate thousands of years of history, resulting in a deeply detailed and dynamic world. The procedural generation in Dwarf Fortress goes beyond mere randomization; it involves creating a coherent and interconnected world where every element, from geography to political structures, is generated algorithmically.

Spore pushed the boundaries of procedural content generation by procedurally generating creatures, planets, and even entire galaxies. Unlike traditional

games, Spore did not store textures, music, or animations. Instead, it generated these elements on-the-fly, allowing for an almost infinite level of variety and creativity.

Minecraft (2011), revolutionized procedural generation in gaming by creating vast, open worlds composed of blocks, with different biomes, caves, and structures all generated procedurally. The game's world is generated using a combination of Perlin noise and other algorithms to create varied and dynamic landscapes that players can explore indefinitely. Minecraft's use of procedural generation allowed for an infinite world size, limited only by the computing power available.

Procedural generation's impact extends beyond gaming into simulations, scientific research, and computer graphics. In scientific simulations, procedural techniques are used to generate terrains for geological studies, allowing researchers to model and analyze various scenarios, such as erosion and sediment transport. Procedural generation is also used in fluid dynamics simulations to create realistic water flows, weather patterns, and other environmental phenomena.

In computer graphics and animation, procedural generation is essential for creating complex visual effects, such as realistic landscapes, textures, and particle systems in movies. Procedural methods enable the efficient generation of vast and detailed environments that would be impractical to model manually. For example, in movies like *The Lord of the Rings* (2001) and *Avatar* (2009), procedural techniques were used to generate large natural landscapes, facial animations, crowds simulations, etc.

Parallelization and GPU utilisation

The exponential increase in computing power over the past few decades allowed exponential advancements in procedural generation, particularly for real-time content creation and complex simulations. The evolution from single-core processors to multi-core CPUs (Central Processing Unit), and more significantly, the rise of powerful Graphics Processing Units (GPUs) capable of massive parallel processing, has transformed how procedural algorithms are implemented.

GPUs are perfect for procedural generation due to their architecture, which allows them to execute thousands of threads in parallel. Unlike CPUs, which are optimized for sequential processing and are typically limited to a few cores, GPUs contain thousands of smaller, simpler cores designed for handling multiple simple tasks simultaneously. This parallelization capability is essential for procedural generation, where large data grids need to be processed efficiently.

In procedural generation, many operations can be performed independently

on different parts of the data, making them ideal candidates for parallel execution. For example, when generating a large terrain, each point on the terrain grid can be computed independently based on noise functions or subdivision algorithms. By distributing these computations across many parallel threads, GPUs can dramatically accelerate the generation process, allowing for real-time updates and interactions in complex environments.

GPGPU (General-purpose Processing on GPU) refers to the use of GPUs for performing computations that are traditionally handled by CPUs. Originally designed for rendering graphics, GPUs have evolved to support general-purpose computing tasks through frameworks like CUDA (Compute Unified Device Architecture) from NVIDIA and OpenCL (Open Computing Language). These frameworks allow developers to write programs that can harness the parallel processing power of GPUs for a wide range of computational tasks beyond just graphics rendering.

In procedural generation, GPGPU is employed to accelerate various algorithms that require intensive computation. For example, noise functions like Perlin or Simplex noise can be computed in parallel across the entire terrain grid or texture space, enabling the real-time generation of complex patterns and surfaces. Or techniques such as the Diamond-Square algorithm or midpoint displacement can be implemented on the GPU, where each refinement step of the grid can be processed simultaneously for different segments of the terrain. Finally, procedural simulations of fluid dynamics, particle systems, and other physical phenomena can be executed on the GPU, where the interactions of thousands or even millions of particles are computed in parallel, resulting in realistic, real-time simulations.

The integration of machine learning, particularly deep learning techniques, with procedural generation represents a the new step forwardin the domain. Machine learning models, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), have been leveraged to enhance the quality and diversity of procedurally generated content.

GANs are composed of two neural networks, a generator and a discriminator, that are trained together in a competitive manner. The generator creates new data samples, while the discriminator evaluates them against real data. Through this process, the generator learns to produce increasingly realistic outputs. In procedural generation, GANs can be trained on datasets of real-world terrains, textures, or architectural styles, and then used to generate new, highly detailed and realistic content that closely mimics the training data. For instance, GANs can create entire landscapes that are indistinguishable from real-world environments, adding a layer of realism that traditional procedural techniques.

VAEs are another type of deep learning model used in procedural generation.

They work by encoding input data into a latent space, where similar inputs are mapped close to each other, and then decoding from this space to generate new data. VAEs are particularly useful for generating content that needs to maintain a certain level of coherence or adhere to specific stylistic constraints. For example, VAEs can generate new character models or textures that fit within a desired aesthetic, providing a high degree of control over the final output.

[TALK ABOUT TRANSFORMERS ALSO]

1.3.3 Terrain generation

Procedural generation encompasses a diverse range of techniques used to create complex, natural-looking content, particularly in terrain generation. These techniques span from mathematical models to advanced simulations and can be broadly categorized into methods for large-scale terrain generation and procedural landform generation.

Noise functions are foundational to procedural terrain generation, providing the basis for creating coherent, natural patterns across terrains. Perlin noise, developed by Ken Perlin in 1983, is a gradient-based noise function that produces smooth, continuous patterns, ideal for generating natural-looking textures and landscapes. Simplex noise, introduced by Perlin in 2001, improves upon Perlin noise by being more computationally efficient and reducing directional artifacts, especially in higher-dimensional spaces. Worley noise, another noise variant, generates patterns based on the distance between points, producing cellular structures useful for simulating natural textures like stone or marble, and for modeling phenomena like cloud formations. Fractional Brownian motion (fBm) builds on these noise functions by summing noise at different scales and amplitudes, creating terrains with varied features, from gentle hills to rugged mountains. Techniques like ridge noise focus on generating sharp features such as crests and ridges, while domain warping applies distortions to prevent grid artifacts and simulate erosion effects without the need for full physical simulations.

Cellular automata offer a rule-based approach to procedural generation, particularly useful for simulating complex systems and natural processes. Originating in the 1940s, these grid-based models evolve based on rules applied to neighboring cells. In terrain generation, cellular automata are commonly employed to simulate cave systems, forests, and other structured environments. Notable examples include Conway's Game of Life, which demonstrates how simple rules can lead to complex, self-organizing patterns, and Langton's Ant, which shows how basic rules can produce intricate behaviors. Cellular automata can be generalized to continuous time and space fields, making them versatile

tools for modeling a wide range of natural phenomena, from fluid dynamics to vegetation patterns.

Large-scale terrain generation involves creating expansive terrains that exhibit natural-looking landscapes over large areas. Techniques like subdivision schemes—including the diamond-square algorithm—iteratively refine an initial coarse terrain by subdividing it and adding fractal detail. While effective in generating self-similar terrains, these methods may introduce artifacts such as directional inconsistencies, which require further refinement. Faulting is another technique, simulating tectonic activity by introducing random vertical faults into a flat terrain. This method creates realistic elevation changes by displacing points on either side of the fault line, with smooth step functions ensuring gradual transitions between displaced and non-displaced areas. Both methods are essential for generating the broad, natural features characteristic of large-scale terrains, although they may lack the precision needed for detailed landform creation.

Procedural landform generation focuses on creating specific terrain features, such as rivers, cliffs, and canyons, with greater precision and control. Controlled subdivision enhances traditional fractal methods by integrating user-defined features, such as river networks, directly into the terrain. This allows for the creation of landscapes that adhere to specific topological and hydrological requirements. Feature-based construction techniques further refine this approach by using interactive sketch-based interfaces, where users can define terrain features through control curves. These curves guide the multi-resolution deformation process, enabling the creation of detailed and complex landforms like layered terraces and isolated mesas. By blending different types of curves, such as elevation profiles and cross-sectional shapes, users can produce terrains that are both realistic and tailored to specific needs.

Neural networks represent a more recent advancement in procedural generation, particularly in terrain and texture generation. Generative models like Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) are used to create realistic terrains and photorealistic images. GANs operate by having two neural networks—a generator and a discriminator—compete to produce outputs that are increasingly realistic. VAEs, on the other hand, encode and decode data to and from a latent space, generating new, similar instances of terrains or textures. These models are powerful tools for adding detail and variety to procedurally generated environments, enhancing the realism and complexity of the final output.

Physical Simulations use models of real-world processes to generate and refine natural features in terrains. These include simulations of fluid dynamics, erosion, sediment transport, and other geological and environmental processes. While these simulations are often used to improve the realism of terrains rather

than for initial creation, they are crucial for adding realistic details such as river erosion, weathering, and sediment deposition. Erosion models mimic natural processes like water flow and wind, shaping the terrain over time, while sediment transport models simulate the movement and deposition of sediment, contributing to the natural evolution of the landscape.

By combining these diverse techniques, procedural generation enables the creation of vast, detailed, and realistic terrains with minimal input data. This approach balances computational efficiency with the complexity required to simulate natural landscapes, making it a powerful tool in terrain modeling and simulation. Whether generating expansive landscapes or intricate landforms, procedural generation techniques provide the flexibility and precision needed to create dynamic and believable environments across various applications, from video games to scientific simulations.

1.3.4 Fluid simulations

Fluid simulations are an essential component of procedural terrain generation, particularly for modeling the behavior of water and its interactions with the terrain. These simulations are crucial for creating realistic and dynamic environmental models that accurately reflect natural phenomena, such as river flow, coastal erosion, sediment transport, and overall hydrology.

At the core of fluid simulations are the Navier-Stokes equations, which describe the motion of fluid substances. These equations account for various forces acting on a fluid, such as pressure, viscosity, and external forces like gravity.

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f} \tag{1.7}$$

Where $\mathbf{u} = (u, v, w)$ is the fluid velocity vector, with components u, v, and w in the x-, y-, and z-directions, respectively, t is time, ρ is the fluid density, p is the pressure field within the fluid, μ is the dynamic viscosity of the fluid, ∇p is the pressure gradient force, $\mu \nabla^2 \mathbf{u}$ is the viscous diffusion term, representing the effects of internal friction within the fluid and finally, $\mathbf{f}$ represents external body forces, such as gravity.

For an incompressible fluid like water, the Navier-Stokes equations can be simplified by assuming that the fluid density remains constant, leading to the incompressibility condition:

$$\nabla \cdot \mathbf{u} = 0 \tag{1.8}$$

This condition ensures that the volume of fluid elements remains unchanged as they move through space, which is crucial for accurately simulating water flow.

The Navier-Stokes equations are highly non-linear, which means that even small changes in the input conditions can lead to significant and unpredictable changes in the fluid's behavior. The non-linearity makes solving these equations a expensive computational problem, as it requires iterative methods to find stable solutions. Each step of the simulation must balance the forces acting on the fluid, such as pressure and viscosity, while ensuring that the solution remains physically accurate and stable over time. This process becomes particularly demanding when aiming for realistic, high-resolution simulations where fine details like turbulence and surface tension must be accurately captured.In this case we require dense computational grids or a large number of particles, and these elements must be updated frequently to maintain the realism of the simulation, which exponentially increase the computations and the memory used.

The computation cost is further amplified when simulations are performed in 3D space. Unlike 2D simulations, where calculations are confined to a plane, 3D simulations must account for the full complexity of fluid movement in all directions. This increases the number of computations required, making real-time applications like as video games and interactive simulations almost incapable to use them.

Different fluid solvers have been developed to approximate the solutions to the Navier-Stokes equations, each with its strengths and weaknesses. These solvers vary in how they represent the fluid (the Lagrangian approach using particles or the Eulerian approach with discrete grids, or a combination of both) and how they handle the computational trade-offs between accuracy, stability, and performance. The choice of a fluid solver depends on the specific requirements of the simulation, such as the need for real-time performance, the level of detail required, or the types of fluid behavior being modeled.

Marker-And-Cell (MAC) Method

The Marker-And-Cell (MAC) method is one of the earliest and most fundamental grid-based techniques for simulating incompressible fluid flows. In the MAC method, the fluid's velocity components are stored at the faces of grid cells, while pressure values are stored at the cell centers. Marker particles are used to track the fluid's free surface, ensuring that fluid interfaces and boundaries are accurately captured. The MAC method excels at maintaining the incompressibility condition of fluids and accurately modeling pressure fields, which are important for realistic fluid dynamics. Because of its emphasis on accuracy and detailed pressure modeling, MAC is more oriented toward realism, especially in scenarios

where the precise behavior of fluids is essential. However, its grid-based nature can lead to challenges in handling complex geometries and fine details at fluid boundaries, as the resolution is limited by the grid size. Moreover, the method can be computationally intensive, especially for high-resolution grids, making it less ideal for real-time applications.

Stable Fluids

Building on the grid-based approach of the MAC method, the Stable Fluids method addresses key stability issues that arise in fluid simulations. Stable Fluids uses a semi-Lagrangian advection scheme and implicit solvers to achieve stability even with large time steps, which is particularly advantageous in real-time applications like video games or interactive simulations. This method is designed with performance in mind, allowing for the simulation of fluid motion without the numerical dissipation that can degrade the accuracy of results over time, which was a common problem in other grid-based methods. While Stable Fluids prioritize performance and stability, particularly in real-time environments, the use of implicit solvers can introduce smoothing effects that reduce the sharpness of fine details in the fluid's motion. Additionally, the method's reliance on grid resolution still limits its ability to represent highly detailed surface interactions, making it a good compromise between realism and performance.

Particle-In-Cell (PIC) Method

The Particle-In-Cell (PIC) method represents a hybrid approach that combines the strengths of particle-based and grid-based methods. Fluid properties are stored on a grid, but the fluid's motion is tracked using particles, which carry velocity and position information only. The particles interact with the grid to update the fluid's velocity field, and the grid provides a stable mean for solving the fluid dynamics equations. PIC is particularly useful for capturing the large-scale motion of fluids, benefiting from the grid's stability while using particles to track detailed fluid behavior. However, the method leans more toward performance over accuracy due to its hybrid nature and ability to handle large-scale fluid movements efficiently. A major downside of PIC is its tendency toward numerical dissipation, where the fluid's kinetic energy is artificially reduced over time, leading to a loss of fine details, especially in turbulent flows. This dissipation occurs because the interpolation between particles and the grid tends to average out small-scale variations in velocity, making PIC less suitable for scenarios where high fidelity and detail are required.

Fluid-Implicit Particle (FLIP) Method

The Fluid-Implicit Particle (FLIP) method is an evolution of the PIC method designed to address its numerical dissipation problem. FLIP retains the grid-particle hybrid approach but modifies how particle velocities are updated. Instead of fully relying on the grid's velocity field, FLIP updates particle velocities by applying only the changes in the grid's velocity field, preserving the fluid's kinetic energy and maintaining the detail in small-scale motions. This approach significantly reduces numerical dissipation, making FLIP more oriented toward realism, particularly in simulations that require detailed fluid dynamics like splashing, swirling, and other turbulent effects. However, FLIP is more computationally expensive than PIC and can suffer from instability issues, particularly when dealing with highly turbulent flows. The increased computational cost and the need for careful tuning of simulation parameters to maintain stability make FLIP less ideal for performance-focused applications but suitable for scenarios where high fidelity and detailed fluid behavior are essential.

Smoothed Particle Hydrodynamics (SPH)

Smoothed Particle Hydrodynamics (SPH) is a purely particle-based method that models fluids using discrete particles, each representing a small volume of fluid. These particles interact with each other based on smoothing kernels, which define the influence of one particle on its neighbors over a certain distance. SPH is particularly well-suited for simulating free-surface flows, such as waves, splashes, and other fluid phenomena where the interaction between fluid particles is complex and highly dynamic. Due to its ability to handle complex boundaries and fluid interfaces naturally, SPH is much more focused on realism, especially in scenarios that require accurate computation of fluid dynamics and interactions. However, SPH is computationally intensive, especially for high-resolution simulations where a large number of particles is required. Additionally, SPH can suffer from stability issues, such as particle clumping or excessive smoothing, which can detract from the realism of the simulation if not carefully managed. These factors make SPH better suited for applications where realism is prioritized over performance, particularly in high-fidelity simulations used in film and scientific research, but not for real-time applications.

1.3.5 User interaction

User interaction affects how users balance realism, speed, and control over generated content.

Realism refers to the extent to which generated content accurately represents real-world characteristics, such as visual details, physical processes, and natural patterns. This is especially important in simulations, visualizations, and training environments where plausibility [FIND BETTER TERM] is critical. Techniques to enhance realism include physical simulations, which model processes like erosion and sediment transport, and the integration of expert knowledge from fields such as geology and ecology. However, achieving realism can be challenging due to the complexity of detailed simulations and the need for specialized expertise, which can demand substantial computational resources and inputs.

Speed is about the efficiency of content generation within acceptable time constraints. While no official categorisation has been set, James Gain proposed to describe levels of speed based on response time:

- Real-time: generation in less than 30 milliseconds, essential for interactive applications like VR environments.
- Interactive: generation in under 3 seconds, suitable for user-driven customization in games and simulations.
- Near-interactive: generation in less than 5 minutes, applicable for larger-scale simulations where some delay is acceptable.
- Long-term: generation that takes longer, often used for precomputed content or offline rendering.

Optimizing speed involves using efficient algorithms and parallel processing. Almost all recent works achieve fast execution time thanks to high parallelisation on GPU. Other types of algorithm may rely on a refinement paradigm, generating a coarse result at first and then iteratively adding finer details, such that the global output shape is available long time before the real final result.

Control refers to how much users can influence or direct the procedural generation process to meet their specific needs or preferences. This includes parameters fine-tuning, allowing users to modify parameters like the noise functions parameters or the simulation parameters, and artistic control tools, made for artists and designers, to guide the generation process with the use of masks and brushes, allowing for combination and mixing of different algorithms.

Managing diverse and sometimes conflicting user expectations, such as balancing creative freedom with the demand for realistic outcomes, requires careful consideration. Additionally, it is essential to balance the complexity of procedural systems to ensure they remain user-friendly, avoiding overwhelming users or producing results that feel artificial or inconsistent.

1.4 Underwater landscapes

- Use "seascape" instead of "landscape" and "aerial landscapes" for above water.
** Not official terms, but may be very useful for the rest of this thesis..!
- Describe main differences with landscape generation
- Describe coral islands ecosystems.

1.4.1 Main differences with aerial terrains

In the field of terrain generation, particularly when it comes to modeling underwater landscapes, the challenge is to accurately represent complex environments that include both geological and biological features. The underwater world is filled with diverse structures, ranging from vast coral reefs to intricate cave systems, each requiring specialized techniques to model their unique characteristics.

3D Data for underwater landscapes

One of the most challenging aspects of underwater landscape modeling is capturing the complexity of its structure. Taking the example of coral reefs,these environments feature highly intricate 3D structures, including branching corals, coral mounds, and encrusting forms. These formations are not just surface-level features; they extend into the terrain with voids and cavities, such as caves, grottos, and karst networks, which add layers of complexity to their representation. Accurate modeling of coral reefs demands that the porous and irregular nature of these formations is represented in high detail, which is a significant challenge due to the variability of their structures.

In addition to biological features, underwater landscapes are shaped by geological processes that must also be accurately represented. For example, sedimentary layers and underwater geological formations like seamounts and ridges are important components of the terrain.

To gather the data needed for such detailed modeling, advanced measurement techniques are employed. Sonar, including multi-beam echo sounders, and LIDAR (Light Detection and Ranging) are commonly used to capture detailed 3D data of underwater terrains. These technologies allow for the mapping of areas that are otherwise difficult to access, providing the foundational data necessary for accurate terrain generation. Remote sensing technologies also play a major role, enabling the collection of data over large and remote areas, further enhancing the model's accuracy, but represent a significant challenge as the approvision

of humans and hardware in these areas is far from easy to achieve.

Interdisciplinary data integration

Accurately modeling underwater landscapes requires more than just geological data; it necessitates an interdisciplinary approach that integrates biological and ecological data. Geological validation is an important step in this process, involving collaboration with geologists and marine scientists to ensure that the models are not only scientifically accurate but also reflective of real-world conditions. This validation process ensures that the generated models accurately depict the geological and biological features present in underwater environments.

Biological data is also essential, particularly when modeling coral reefs and other marine ecosystems. Incorporating data on coral species, marine biodiversity, and ecosystem dynamics allows for the creation of realistic and biologically accurate representations. Additionally, the impact of biological processes, such as coral growth patterns and marine erosion, must be considered, as these factors significantly influence the terrain shape over time.

However, one of the major challenges in modeling underwater landscapes is the scarcity of high-resolution data, especially in remote, protected or less studied areas. This data scarcity makes it difficult to create detailed and accurate models. Another challenge lies in the integration of diverse data types, geological, biological, hydrological, etc, into a cohesive and comprehensive model. Effective integration requires complex techniques to ensure that all aspects of the underwater environment are accurately represented.

Multi-scale modeling

Underwater landscapes are characterized by features that vary greatly in scale, from vast underwater mountains and ridges to small-scale elements like individual coral polyps and marine vegetation. This diversity in scale necessitates multi-scale modeling techniques that can accommodate both large and small features within the same model.

A. Paris [CITATION THESIS] classified the different geological structures in four different spatial scales: the *megascale* describes landforms spread on more than 100km such as continents or mountain ranges, the *macroscale* between 1km and 100km like river or cave networks, the *mesoscale* ranging from 10m to 1km contains most complex structures (arches, ravines, cliffs, ...) and finally the *microscale* for features between 10cm and 10m, like sand ripples, ventifacts, rocks, etc. Generally, we consider as *large-scale* features that can be seen from far away and *small-scale* elements for which somebody has to be to close to to observe it.

Large-scale features, such as underwater mountains and ridges, define the macrostructure of the landscape. These must be integrated seamlessly with small-scale features, such as detailed sedimentary textures and small coral formations, to ensure that the terrain is represented accurately across different scales.

[TODO]

The procedural generation of underwater landscapes involves the integration of advanced 3D data collection techniques, interdisciplinary collaboration, and multi-scale modeling approaches. Modeling underwater environments, from the complex structures of coral reefs to the accurate representation of geological and biological features, requires innovative solutions that combine geological plausibility with visual realism.

1.4.2 Coral reefs (biological aspects)

[BLA BLA BLA TODO]

1.5 Prototype creation

- C++23 and Qt5.12
- OpenGL 4.6 and GLSL
- Marching Cubes on geometry shader
** Bad idea, but justify why
- Renderings:
** With the prototype:
*** Marching Cubes on geometry shader
*** Triplanar texture
*** Real-time results
*** Textures based on materials
** With Unreal Engine 5:
*** Static meshes
*** Added procedural vegetation with plugin [PLUGIN NAME] and ocean with [PLUGIN NAME]
** With Blender 4.1:
*** Static meshes
*** Easier script usage
- ...

Part I

Semantic Representation

Abstract

This chapter introduces a novel method for procedural terrain generation, which leverages a sparse representation of environmental features to produce landscapes that are lightweight, plausible and adaptable to user desires. The method differs from traditional terrain generation approaches by emphasizing multi-scale user interaction and incorporating expert knowledge to model the evolution of terrain features over time. By representing terrain features as discrete entities, or "semantic terrain entities", the method enables dynamic interaction between these entities and their surrounding environment, represented through continuous scalar and vector fields. The generation process is iterative and allows for user-guided modifications at any iteration, including the introduction of environmental events that can influence the terrain's evolution. The proposed approach is particularly flexible, capable of generating both terrestrial and underwater landscapes with a focus on large-scale plausibility and detailed, localized feature representation.

CHAPTER 2

Generation de terrain sémantique

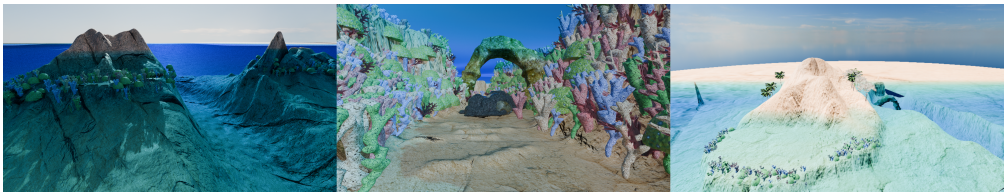


Figure 2.1: Our method can produce different scenes including coral islands and canyons at multiple scales using semantic terrain entities to represent terrain features.

Contents

Back to summary

2.1 Introduction

Topographic maps are very useful tools for biologists, geologists or even oceanologists (which would call them "bathymetric charts"). These maps are displayed in 2D but provide 3D information about the altitude (or depth), but can also use symbology to represent the important elements that need to be visible. Map symbols are important in order to extract as much information as possible from a 2D object. In cartography, map symbols are defined as geometric primitives such as points, polylines, polygons, and (rarely) polyhedrons. These symbols are a simplification of the content of an environment, or an abstraction of the 3D shape of the terrain features. This is useful to understand the relationship between the different features, which may enable to deduce physical rules in the evolution of a terrain.

In such way, geologists can study the distribution of peaks in a mountain range, the location of soil types in an area, which in turn allow to deduce possible locations of karst networks, for example. Using the same tools, a biologist may interpret the effect of natural or artificaial reefs on coastal erosion, or understand better the interactions inside an ecosystem. Oceanologists may deduce the formation of canyons and fans from old river systems.

By simplifying the surface details in maps or models, we can concentrate more effectively on gaining a deep understanding of the underlying processes that shape the terrain. This focused understanding allows us to apply the insights we gain to a wide variety of terrain types. Essentially, this approach enables us to generalize our findings and apply them across diverse geographical landscapes, facilitating broader and more versatile applications of our knowledge.

Parallelly, terrain artists most of the time sketch the global shape of the terrain they will model beforehand, such that they can check, before the modeling part, that the consistency and plausibility of the terrain will be valid. Looking at a

simplified map before starting the modeling step allows the designer to modify the overall shape of the terrain, at a large scale, before the 3D geometry comes into play, generating too much control points or vertices to be able to deal with.

Starting from an initial configuration or providing conditions on the desired output terrain, the algorithm we propose will let the different terrain features evolve as a multi-agent system in which the user can apply modifications or new constraints on the state of the environment. The resulting configuration is an environment conform to the constraints given by the user over the distribution of features present in the scene.

As described in the previous chapter, most terrain generation algorithm use the geometry of an initial terrain surface to iteratively apply changes such as noise-based algorithms or erosion simulation. While the introduction of information about some environment variables or properties of the ground may influence the result of an algorithm, the control of the global shape of the terrain is lost as it treat locally the surface, without knowing which feature a certain point on the surface lies on.

The question which led to our solution is the multi-scale user interaction: "Is it possible to provide an interaction mean for terrain generation allowing the user to interact with a small structure like a rock in the same manner as with a large structure like a mountain?". In this work, we want the user to be able to have a large scale representation of the terrain in order to generate a landscape that satisfies his needs while keeping the possibility to apply large modifications. In discussion with robotician users, we realise that we want to create a large landscape that can contain interesting configurations, select a smaller region that have features in a disposition that fits its requirements and then refine again the given region. In the optic of generating a large scale terrain in which we could focus the generation effort in a certain region, we wished to be able to see a coarse representation that can be computed quickly.

We aim for our terrain generation method to be versatile enough to handle both terrestrial and aquatic environments. This dual capability would allow the method to be applicable to landscapes above the water level, such as mountains and valleys, as well as to submerged terrains, such as oceanic canyons and coral landscapes.

Because many geographical terms and computer science terms are deceptive cognates, we will try to find middle ground in the naming of our introduced structures to avoid as much as possible any ambiguity between the research fields.

2.2 Related works

Procedural terrain generation has been heavily studied for the last 40 years (E. Galin et al., 2019). Researches in this topic try to find new solutions to compromise between realism, user control and efficiency (Gain et al., 2009). Using fractal noise parametrized to resemble real landscape has been an important first step (Musgrave et al., 1989) as it's a fast and light solution to generate procedurally the appearance of mountains. The lack of user control pushed newer works toward the use of controlled noise by including real DEM in the process through learning (Brosz et al., 2007; Kapp et al., 2020), while the rise of deep learning technologies gave higher control to the user through sketches (É. Guérin et al., 2017; Talgorn and Belhadj, 2018).

By including expert knowledge of tectonic process and subsurface geology, some algorithms tend to get more realistic (Cortial et al., 2019; Michel et al., 2015; Patel et al., 2021).

While these algorithms are able to generate large-scale landscapes, the finer details of the terrain is often computed by the use of erosion simulation (Cordonnier et al., 2023; Paris et al., 2019b; Schott et al., 2023). This process can be expensive in time but results in more plausible surfaces.

All the algorithms aim to reproduce plausible relief in terrestrial landscapes, mostly limited to alpine landscapes, but a lack of research can be found in almost all other biomes. Underwater landscapes generation, for example, has been almost completely absent from literature for many reasons: the difficulty of accessing the area, the lack of visibility under water and the complex physics of underwater geology and biology make the algorithms adapted for this environment scarce.

The majority of the ocean floor can be represented as a fractal terrain (Mareschal, 1989). While stochastic noise can be sufficient to model the ocean floor, this process won't cover areas with the biggest biomass, near shallower waters such as near coasts and islands.

Due to the impossibility to observe the large-scale and the small-scale of underwater environments, some works related to geology model large structures like the profile shape of the coral reef (Bosscher and Schlager, 1992), simulate its surface growth (Li, 2021), or use procedural algorithms for single coral colonies (Abela et al., 2015). We however don't have a mix of the different scales, and neither methods take into account the environment such as the topography or the interaction of different terrain features. This is mainly due to the fact that the evolution time for each scale varies from a span of weeks to thousands of years.

In an ecosystem, every element within the system has an impact on its surroundings, and accurately simulating physical properties like shading, heat, and

humidity can require immense computational power. To address this, instead of simulating these properties individually for each object, they can be represented as scalar fields that span the entire scene. These scalar fields, which may represent temperature, humidity, or occlusion, are locally influenced by nearby objects and terrain features, such as trees casting shadows or buildings radiating heat. This approach, as described in (Grosbellet et al., 2016; É. Guérin et al., 2016), allows for the environment's physical properties to be computed in a more efficient manner. By simplifying the complex interactions into these localized scalar fields, their method provides a scalable system that can handle large and detailed scenes. This system effectively renders scene details, such as snow accumulation or leaf distribution, in a way that is visually plausible, ensuring realistic results without the need for exhaustive simulations. In a similar approach, Wejchert and Haumann (Wejchert and Haumann, 1991) represent wind flow as a composition of local vector fields, known as "flow primitives", which can be combined to simulate complex fluid behavior. This method simplifies the computation by avoiding full fluid simulations, instead offering a lightweight model that allows for intuitive user control over the flow dynamics. By combining these approaches, we can create dynamic ecosystems where environmental effects and fluid flows interact in a plausible manner, while requiring a low computation effort and preserving a high user control.

2.3 Description of the method

In this section, we present the pipeline and processes that underpin our method. Our approach introduces the concept of semantic terrain entities, simplified terrain features that interact with their surroundings to simulate complex ecosystem dynamics. These semantic terrain entities, which can represent natural features such as trees, rivers, or rocks, influence and are influenced by scalar fields like temperature, humidity, and elevation. The method is structured into several key phases: initialization, where the foundational elements of the terrain and semantic terrain entities are set up; an iterative generation process, where these semantic terrain entities are instantiated and interact with their environment; and finally, the production of a sparse representation of the scene's features. This section details each of these phases, explaining how the system dynamically adapts to user input and environmental changes.

2.3.1 Pipeline

Initialization

The generation of the terrain is initialized using an initial height field h and an initial water level $\mathcal{L}$. During this chapter we will include many features depending on altitude or depth, so we will use the shorthand notations $\mathcal{H} = h - \mathcal{L}$ and $\mathcal{D} = -\mathcal{H}$. The height field provides variation on the altitude, which can influence the generation process of the scene.

The list of available semantic terrain entities $\tilde{\mathcal{S}}$, representing the different features that can be present in the scene, are provided with their properties: type, size, generation rules and effects on the environmental attributes (Section 2.3.2). A target list of semantic terrain entities $\hat{\mathcal{S}}$ can be defined to control the final result of the generation.

Finally, different environmental materials are defined with their properties such as diffusion speed, mass, decay rate and influence from the water currents. An initial environment configuration resulting from the initial height field $\mathcal{H}$ and water level $\mathcal{L}$, the environmental materials distribution $\mathcal{M}$ (represented as scalar fields $\mathcal{M} : \mathbb{R}^2 \mapsto \mathbb{R}$) and water currents $\mathcal{W}$ (as a vector field $\mathcal{W} : \mathbb{R}^2 \mapsto \mathbb{R}^2$) will be used by the semantic terrain entities of the scene to simulate their growth and spawn at the most probable position. The environmental attributes are noted $\mathcal{E} = (\mathcal{D}, \mathcal{W}, \mathcal{L}, \mathcal{M})$ (Section 2.3.3).

The definition of semantic terrain entities' properties and environmental attributes is done with field experts, providing the pertinent parameters required to model the evolution of the terrain features using expert knowledge (Section **??**).

The generation phase starts with an initial set of semantic terrain entities present in the scene, which can optionally be pre-filled.

Generation process

Once the initialization phase is done, the generation begins. The generation process is incremental and its main loop is composed of two different steps: the instantiation of new semantic terrain entities then the update of the environment.

Instantiation

At each iteration, new semantic terrain entities can be created at their most fitting locations if possible. The generation rules provided in the initialization phase are used to find an optimal position from stochastic sampling (Section 2.4). All semantic terrain entities are evaluating their state analytically using a fitness function and a skeleton fitting function provided as input (Section 2.4).

Environment update

Once the instantiation step is done, the environmental attributes are updated by each semantic terrain entity through environmental modifiers, which depose and absorb some of the environmental materials $\mathcal{M}$ (Section 2.5) while modifying the water currents $\mathcal{W}$ (Section 2.5.3) and the height field $\mathcal{H}$ around them. Finally, water currents and terrain slope displace environmental materials of the terrain until reaching a dynamic equilibrium in the environment at each iteration.

During the generation process, the user can alter directly the distribution and shapes of the semantic terrain entities (Section 2.6.1) and perturb the generation process by planning geomorphic events that have impacts on the environmental attributes (Section 2.6.2).

Output

The output of our system is a set of semantic terrain entities disposed in the plane. We do not provide the 3D representation of the semantic terrain entities in this chapter, letting the user define the rendering method. The figures used in the chapter use a mix of implicit surfaces and triangular meshes.

2.3.2 Semantic terrain entities $\mathcal{S}$

A geographical feature, also called object or entity, is defined as a discrete phenomenon located at or near the Earth's surface, relevant in geography and geographic information science (GIScience). It represents geographic information that can be depicted in maps, geographic information systems (GIS), and other forms of geographic media. This term includes both natural and human-made objects, ranging from tangible items like buildings or trees to intangible concepts like neighborhoods or savana. Features are distinct entities with defined boundaries, differentiating them from continuous geographic masses or processes occurring over time. They can be categorized as natural features, such as ecosystems, biomes, water bodies, and landforms, or artificial features, such as settlements, administrative regions, and engineered constructs. Geographic features are described by characteristics including identity, existence, classification, relationships with other features, location, attributes, and temporal aspects. Information about these features is stored in geographic databases using models like GIS datasets, which organize and represent these features in structured formats. As the term "feature" is overused in computer science, we will use the term semantic terrain entity in this work.

Each semantic terrain entity is shaped with a simple geometric shape called a "skeleton" that defines where it is located and how it fits into the environment. The skeleton can either be, as used in cartography, a point, a curve or a region. We will then refer to our semantic terrain entities as point-based, curve-based

or region-based, respectively. These semantic terrain entities interact with the environment $\mathcal{E}$ by changing local conditions using the environmental modifiers. For example, the presence of a river might increase the moisture in the surrounding area, while a mountain might induce more rockiness in the soil composition. They can also absorb changes from the environment, such as a forest taking in humidity from the air. The placement of semantic terrain entity is determined by a fitness function ω, which evaluates how suitable a location is based on the environmental attributes. Once a suitable location is found, the skeleton fitting function optimizes the shape and position of the entity to fit as best as possible into the environment Γ.

2.3.3 Environmental attributes $\mathcal{E}$

In geography, a "field" refers to a continuous spatial phenomenon across a region where each point has a specific value of a variable, unlike discrete objects with distinct boundaries. Fields can be scalar, representing a single value at every point (like temperature or elevation), or vector, representing quantities with magnitude and direction (like wind velocity). Examples include topographic fields for elevation, climatic fields for temperature or precipitation, and magnetic fields for magnetic forces. Because of the ambiguous nature of the term "field" with mathematics and computer science, we will define the geographic fields as environmental attribute.

In an ecosystem simulation, each actor of the ecosystem has an impact on all other actors, which results in an exponentially growing computation effort as the number of elements of the terrain increase. We avoid this problem by considering the environmental attributes as a proxy to allow any semantic terrain entity to interact with any other one. Each of the semantic terrain entity have a local impact on the environmental attributes without knowledge of neighboring semantic terrain entities. This modification of the environmental attributes are presented as the effect of environmental modifiers defined for each semantic terrain entity.

In this work, we have integrated vector environmental attributes (e.g., water currents $\mathcal{W}$) and scalar environmental attributes (e.g., altitude $\mathcal{H}$, water level $\mathcal{L}$, and various material properties $\mathcal{M}$) under the unified term "environment," denoted as $\mathcal{E} = (\mathcal{H}, \mathcal{W}, \mathcal{L}, \mathcal{M})$. The environmental materials represent abstract quantities such as the availability of sand, salt, moisture, or rocks at each point. It is important to emphasize that these materials are not to be visualized as physical layers stacked on the terrain surface. Instead, they should be understood as conceptual resource distributions that influence the environment and the behavior of semantic terrain entities, rather than as something directly observable

in the scene.

Environmental modifiers $\mathcal{E}^+$

The environment determine if a semantic terrain entity does belong at a certain position. When a semantic terrain entity is placed, its surrounding environmental attributes can be affected though environmental modifiers noted $\mathcal{E}^+ = (\mathcal{H}^+, \mathcal{W}^+, \emptyset, \mathcal{M}^+)$ defining a change of height $\mathcal{H}^+$, changes in the water currents $\mathcal{W}^+$ and environmental material alteration $\mathcal{M}^+$.

Semantic terrain entities are subject to altitude conditions. However, to maintain a clear distinction between semantic modeling and 3D modeling, we do not compute the exact physical shape or detailed height field of each semantic terrain entity. Instead, we define a coarse height function, a parametric representation derived from the skeleton to provide a rough estimate of the changes in elevation around it. This simplified model of how the semantic terrain entity influences the surrounding terrain's altitude allows us to cheaply evaluate the potential presence of new entities that have altitude-dependent conditions without the need to perform expensive computations to generate a detailed height map of the entire terrain.

Altering the vector field of the water currents $\mathcal{W}$ is done by the composition of the effect $\mathcal{W}^+$ of each object at a position $\mathbf{p}$ as introduced in Wejchert and Haumann, *Animation aerodynamics* (1991) , while we use the formulation of Kelvinlets (Goes and James, 2017) in the computation of effect of each semantic terrain entity.

Each semantic terrain entity has intrinsic environmental materials that can be seen as "spreading" and "absorbed" around its skeleton over time. A coral reef may produce coral polyps and at the same time reduce the water currents. It grows thanks to the deposition of limestone from coral colonies. In our model, the colonies affect the environmental attributes through the deposition of environmental material $\mathcal{M}^+_{\text{limestone}}$, which in turn, is absorbed by the coral reef, without a direct exchange between the two semantic terrain entities.

The alteration of a scalar environmental attribute is done by adding or removing some amount around the skeleton of the semantic terrain entity and diffusing it in the space, influenced by the water currents. We consider the system to be steady state, garantied by the introduction of a decay rate $k > 0$ in the computation of the diffusion and advection.

2.4 Placement of semantic terrain entities in an environment

At each iteration of our algorithm, we want our semantic terrain entities to be at plausible positions. We do not guaranty a temporal continuity between iterations as in Ecormier-Nocca et al., *Authoring consistent landscapes with flora and fauna* (2021) , so the objective is to add new semantic terrain entity in order to satisfy the users wishes, while conserving the plausibility of the scene. Rather than "forming" these semantic terrain entities, our method "reveals" them, much like a paleontologist uncovers fossils during an excavation. A paleontologist does not dig randomly across the Earth to find fossils; instead, they analyze the geological context to identify the most likely locations. Similarly, our method observes the environment and estimates where certain elements are likely to exist. For example, in hydrology, if a river appears to originate from nowhere, it might suggest the presence of a karstic river system upstream. In urban planning, if many roads converge at a certain point, it is reasonable to expect that a city is located there. This approach ensures that Semantic Terrain Entities are revealed in positions that are contextually appropriate and coherent within the landscape.

We will follow the same intuition using a fitness function for each of the semantic terrain entity that may be spawn in the terrain. The fitness function defined $\omega : \mathcal{E} \mapsto \mathbb{R}$ provides a score indicating how well the semantic terrain entity may fit in this position. Evaluating this function at multiple position results in an approximation of the fitness map of the entity. Once the most probable position is found, we can find the most plausible shape of the semantic terrain entity using the skeleton fitting function Γ.

For this task, we place a new elements required at the most plausible position using the analysis of the fitness function of each semantic terrain entity. We know that each semantic terrain entity will modify the environment surrounding, which may make previously instantiated semantic terrain entities unfitted. Knowing this,the goal is to add the new element at the position that will change the least the stability of the system.

Genetic algorithms or Depth First Search algorithms could be used to try many possibilities until a local or global minimum could be found, but this would require a large processing power. Naive genetic algorithms would place a semantic terrain entity at a certain position at each iteration and evaluate the stability of the environment, repreating this operation while varying slightly the position of the semantic terrain entities or the type of semantic terrain entity instantiated at each iteration, resulting in way too much computation to stay interactive. The Depth First Seach algorithms would require to compute all the possible combinations of semantic terrain entities and positions which, given

the fact that we want a continuous position in order to work multi-scale, would require to compute an incredibly high amount of possible configurations in order to find a plausible situation, on average. We will work with an evolutionary algorithm to find a compromise between fast computation and a satisfying result.

Our placing algorithm is done in two steps: first, it identifies the global location where a semantic terrain entity best fits within the environment $\mathcal{E}$ using its fitness function ω, which evaluates the suitability of each point $\mathbf{p}$ based on the environmental attributes $\mathcal{E}_p$, including factors such as altitude $\mathcal{H}(\mathbf{p})$, water current velocity and direction $\mathcal{W}(\mathbf{p})$, water level $\mathcal{L}(\mathbf{p})$, and the availability of environmental material $\mathcal{M}(\mathbf{p})$. Second, once the most suitable location is identified, the algorithm determines the most plausible shape of the semantic terrain entity using the skeleton fitting function Γ.

To ease the reading the functions $\omega(\mathcal{E}(\mathbf{p}))$ and $\Gamma(\mathcal{E}(\mathbf{p}))$ with $\mathcal{E}(\mathbf{p})$ the environmental attributes at the point $\mathbf{p}$ of the terrain are simplified to $\omega(\mathbf{p})$ and $\Gamma(\mathbf{p})$ respectively.

2.4.1 Fitness function ω

"Darwinian fitness" refers to an organism's ability to survive and reproduce in its environment. It is a measure of how well-suited an organism is to its surroundings, and those with higher fitness are more likely to pass on their genes to the next generation. In a similar vein, the fitness function of a semantic terrain entity evaluates its suitability at a location in the terrain, extending the meaning to living features like forests and corals and non-living elements such as rivers, mountains, karsts, etc.

The fitness function is constructed by evaluating several environmental variables at a given location $\mathbf{p}$. These include altitude ($\mathcal{H}(\mathbf{p})$) and its gradient ($\nabla\mathcal{H}(\mathbf{p})$), the availability and gradient of various materials ($\mathcal{M}_i(\mathbf{p})$ and $\nabla\mathcal{M}_i(\mathbf{p})$), water current characteristics ($\mathcal{W}(\mathbf{p})$), and water level ($\mathcal{L}(\mathbf{p})$). Altitude and its gradient can influence the placement of objects like rivers, which may prefer lower elevations, or forests, which might thrive on slopes. Each material, such as limestone, sand, or clay, is considered separately, and their availability and gradients at a location are crucial for determining the suitability of different semantic terrain entities, such as a coral reef requiring specific substrates. The velocity and direction of water currents are essential for placing aquatic features or determining where erosion might occur, while the proximity to water influences the likelihood of placing wetlands, lakes, or other hydrological features.

These environmental variables are typically combined in the fitness function, often expressed as a weighted sum, but not restricted to this form:

$$\omega(\mathbf{p}) = w_1 \cdot \mathcal{H}(\mathbf{p}) + w_2 \cdot \nabla\mathcal{H}(\mathbf{p}) + \sum_i (w_{3,i} \cdot \mathcal{M}_i(\mathbf{p}) + w_{4,i} \cdot \nabla\mathcal{M}_i(\mathbf{p})) + w_5 \cdot \mathcal{W}(\mathbf{p}) + w_6 \cdot \mathcal{L}(\mathbf{p}$$

Here, $w_1, w_2, w_{3,i}, w_{4,i}, w_5, w_6$ are weights that reflect the relative importance of each factor for the specific semantic terrain entity being considered.

Different types of semantic terrain entities require different criteria for their fitness evaluation. For example, a river might prioritize lower altitude and proximity to water currents, while a forest might prioritize higher altitude and specific material availability. The flexibility of the fitness function allows it to be customized for each semantic terrain entity, ensuring that the generated terrain remains coherent and realistic.

2.4.2 Skeleton fitting function Γ

The seed point of a spawning semantic terrain entity is defined at the point $\mathbf{p}$ satisfying $\arg\max_{\mathbf{p}} \omega(\mathbf{p})$.

Point-based skeleton

The spawning position of a punctual semantic terrain entity is found at the local maxima of the skeleton fitting function from the seed point. While the skeleton fitting function doesn't have to be identical to the fitness function, we usually use $\Gamma = \omega$ for point-based semantic terrain entities. If the two functions are different, the optimisation process simply follows the field's gradient $\nabla\Gamma$ until the local maxima is reached.

Curve-based skeleton

A curve can have different generation rules, making the definition of the functions easier depending if the semantic terrain entity's skeleton has a notion of direction. It can either:

- follow the gradient of the skeleton fitting function $\nabla\Gamma$, useful when the direction of the skeleton has an importance like a river that has to run downhill,
- or optimize the energy of the function over its curve, like a coral reef that has to cover as much coral colonies as possible.

Following the gradient

Following the gradient of the skeleton fitting function $\nabla\Gamma$ is done by extending a curve from the seed point towards $\nabla\Gamma$ until reaching a local maximum while also following $-\nabla\Gamma$ until finding a local minimum. We stop the building process when our curve reaches a length limit L.

This skeleton fitting function strategy is efficient to describe rivers as we can set $\Gamma(\mathbf{p}) = \mathcal{H}(\mathbf{p})$ to force it to always run downstream.

Energy optimization

Using a modified version the Active Contours algorithm (Kass et al., 1988), we can optimize the energy for the curve C given $E = E_{\text{internal}} + E_{\text{shape}} + E_{\text{image}}$ with

$$E_{\text{internal}} = \frac{1}{2}\left(\left\|\frac{dv}{dt}(t)\right\|^2 + \left\|\frac{d^2v}{dt^2}(t)\right\|^2\right) \tag{2.1}$$

$$E_{\text{shape}} = \left(L - \int_0^1 C'(t)\,dt\right)^2 \tag{2.2}$$

$$E_{\text{image}} = \int_{\mathbf{p}\in C} \Gamma(\mathbf{p})\,d\mathbf{p} \tag{2.3}$$

The internal energy E_{internal} force the shape continuity while the shape energy E_{shape} force the shape into a specific target length L, and finally E_{image} that use the definition of the skeleton fitting function to attract the points of the curve toward a local minimum.

As we allow control over control points of the curve, we approximate this energy in the discrete form using:

$$E_{\text{internal}} = \frac{1}{2}\left(\left\|\frac{dv}{dt}(t)\right\|^2 + \left\|\frac{d^2v}{dt^2}(t)\right\|^2\right) \tag{2.4}$$

$$E_{\text{shape}} = \left(L - \int_0^1 C'(t)\,dt\right)^2 \tag{2.5}$$

$$E_{\text{image}} = \int_{\mathbf{p}\in C} \Gamma(\mathbf{p})\,d\mathbf{p} \tag{2.6}$$

Region-based skeleton

A region is defined as an isocontour of the field for which the target area A is found. From the seed point, we follow the isolevel of the fitness function $\nabla\Gamma^{\perp}$ until a loop is created to define the initial condition of the shape. Using the Active Contours algorithm, we can optimize the region's energy defined as

$E = E_{\text{internal}} + \gamma E_{\text{shape}}$ with

$$E_{\text{internal}} = \frac{1}{2}\left(\alpha(t)\left\|\frac{dv}{dt}(t)\right\|^2 + \beta(t)\left\|\frac{d^2v}{dt^2}(t)\right\|^2\right). \tag{2.7}$$

The internal energy E_{internal} force the shape continuity while the shape energy E_{shape} force the shape into a specific target. For the semantic terrain entities generated in the following examples, we used a constraint on a target area.

$$E_{\text{shape}} = (A - a)^2 \tag{2.8}$$

with a the current area of the shape and A the target area, provided by the user for each shape, with some randomness.

Semantic terrain entities are evaluated at every iteration in order to determine if they are still fitted to belong at the same position as the previous iteration. For punctual semantic terrain entities, this evaluation is applied at its position $\Gamma = f(\mathcal{E}_p)$. Curve semantic terrain entities are evaluated along the parametric curve C such that $\Gamma = \int_C f(\mathcal{E}_{C(t)})\,dt$. In practice, we compute the evaluation as the average of all control points of the curve $\frac{1}{n}\sum_i^n f(\mathcal{E}_{C_i})$. Region semantic terrain entities are evaluated inside their region Ω as such $\Gamma = \int_\Omega f(\mathcal{E}_p)\,dp$. In practice, we compute the average of random points in the region $\frac{1}{n}\sum_i^n f(\mathcal{E}_{p_i})$. When deformations on the semantic terrain entity's shape is applied, we apply cage deformation using Green's coordinates in order to keep consistent evaluation points during the whole life cycle of an semantic terrain entity.

[EXPLAIN that alive = $(\omega > T_\omega) * (\Gamma > T_\Gamma)$ with T_ω, T_Γ user-defined thresholds]

2.5 Environmental modifiers

2.5.1 Environmental material modifiers

The environment is composed of a scalar field for each of the possible material that can be found in the terrain. The scalar fields represents the availability of the material at any point, but not a height field. Each material is defined with a mass m, a fluid velocity factor ν, a diffusion rate D and finally a decay rate k.

Each semantic terrain entity in the terrain is a source and a sink of environmental materials. It is the main mean of communication between semantic terrain entities as it allows them to interact with their surrounding environment. We define the amount of deposed material with $D_{\mathcal{M}}$ and $A_{\mathcal{M}}$ the amount of material deposed and absorbed by the semantic terrain entity and $\gamma(t) \in [0, 1]$ a

factor related with the current state of the semantic terrain entity, which state that more material will be displaced when the semantic terrain entity is fully formed than when it was just spawn:

$$\int_0^t \gamma(t)\,(D_{\mathcal{M}} - A_{\mathcal{M}})\;dt$$

The deposition and absorption around an semantic terrain entity is defined using the Gaussian kernel distance computation from the skeleton.

The scalar field for the material $\mathcal{M}$ is displaced by using a warp operator Φ, taking into account the water flow $\mathcal{W}$ and the terrain slope $\nabla\mathcal{H}$. We unified the warp with m the mass of the material and ν a influence factor of the fluid on the material:

$$\Phi(\mathbf{p},t) = m\nabla\mathcal{H}(\mathbf{p},t) + \nu\mathcal{W}(\mathbf{p},t)$$

The environmental materials are also dispersed at a diffusion rate D, for which we can use the advection-diffusion-reaction equation to evaluate the distribution after a time t

$$\frac{\partial\mathcal{M}}{\partial t}\Phi\nabla\mathcal{M} = D\nabla^2\mathcal{M} - k\mathcal{M} \tag{2.9}$$

We solve Equation (2.9) numerically using Euler integration

$$\begin{aligned}\mathcal{M}(\mathbf{p},t+dt) = \mathcal{M}(\mathbf{p},t) + dt(D\nabla^2\mathcal{M}(\mathbf{p},t) - k\mathcal{M}(\mathbf{p},t) \\ - \Phi(\mathbf{p},t)\nabla\mathcal{M}(\mathbf{p},t))\end{aligned} \tag{2.10}$$

The introduction of the decay rate $k \in]0;1]$ in the equation allows for the reach of a steady-state, where we can consider the simulation stable. As the user updates the state of the simulation manually, we observe the reach of this steady state before continuing the iterative steps.

2.5.2 Height modifiers

The computation of the height $\mathcal{H}(\mathbf{p})$ is done using the coarse height function of each semantic terrain entity affecting a point $\mathbf{p}$.

2.5.3 Influence on water currents

We define our water currents as a vector field defined as

$$\mathcal{W}(\mathbf{p}) = \mathcal{W}_{\text{user}}(\mathbf{p}) + \mathcal{W}_{\text{simulation}}(\mathbf{p}) + \mathcal{W}^{+}_{\text{entities}}(\mathbf{p})$$

With $\mathcal{W}_{\text{user}}$ a user-defined vector field, $\mathcal{W}_{\text{simulation}}$ an analytical solution inspired by a wind flow simulation (Paris et al., 2019a), and $\mathcal{W}^{+}_{\text{entities}}$ the water flow alteration computed from the semantic terrain entities. The component $\mathcal{W}_{\text{simulation}}$ is terrain-induced. Given an input flow direction a, we modify the vector field by warping it with the terrain gradient smoothed at multiple scales :

$$\mathcal{W}_{\text{simulation}}(\mathbf{p}) = \sum_{i=0}^{i=n} c_i \Phi_i \cdot v$$

with $v = (a(1 + k_w \mathcal{D}(\mathbf{p}))$ and $\mathcal{D}(\mathbf{p})$ the depth at point $\mathbf{p}$ and k_w a scaling factor, used to simulate the Venturi effects. $\Phi_i \cdot v$ is the warping operator at scale i with a coefficient c_i defined as

$$\Phi_i \cdot v = (1 - \alpha)v + \alpha k_i \nabla \tilde{h_i}^{\perp}(\mathbf{p}) \qquad\qquad \alpha = \left\| \nabla \tilde{h_i}(\mathbf{p}) \right\|$$

with k_i a deviation coefficient, α the slope of the smoothed terrain and $\nabla \tilde{h_i}^{\perp}(\mathbf{p})$ the orthogonal vector of the smoothed terrain. As proposed by the authors, we used two scaling levels $n = 2$ with gaussian kernels of radii 200m and 50m with weights 0.8 and 0.2 and deviation coefficients k_0 and k_1 of 30 and 5.

$\mathcal{W}^{+}_{\text{entities}}$ is a deformation field defined as the accumulation of flow primitives (Wejchert and Haumann, 1991). Kelvinlets are applied on each semantic terrain entities to deflect the water flow. We use the scale and grab formulations of the regularized Kelvinlets brushes (Goes and James, 2017), denoted as $s_\varepsilon(r)$ and $g_\varepsilon(r)$ respectively to simulate obstruction and diversion, are defined as

$$s_\varepsilon(r) = (2b - a)\left(\frac{1}{r_\varepsilon^3} + \frac{1}{2r_\varepsilon^5}\right)(sr)$$
$$g_\varepsilon(r) = \left[\frac{a - b)}{r_\varepsilon} I + \frac{b}{r_\varepsilon^3} r r^t + \frac{a\varepsilon^2}{2r_\varepsilon^3}\mathbf{I}\right]\mathbf{F}$$

with $a = \frac{1}{4\pi\mu}$ and $b = \frac{a}{4(1-v)}$ provided μ a shear modulus and v a Poisson ratio provided for each Kelvinlet, $r = \mathbf{p} - \mathbf{q}$ for $\mathbf{p}$ the evaluation position and $\mathbf{q}$ the center point of the Kelvinlet, $r_\varepsilon = \sqrt{\|r\|^2 + \varepsilon^2}$ the regularized distance, ε a radial scale for the deformation field, s a scaling factor and $\mathbf{F}$ the force vector of the grab operation. Deformations defined on curves use $\mathbf{q} = C(\mathbf{p})$ with $C(\mathbf{p})$ the closest point on the curve from the point $\mathbf{p}$ and $f = C'(\mathbf{p})$. We can then define $u_o(\mathbf{p}) = s_\varepsilon(\mathbf{q} - \mathbf{p}) + g_\varepsilon(\mathbf{p} - \mathbf{q})$. Finally, we can retrieve the velocity field from the objects:

$$\mathcal{W}^{+}_{\text{entities}}(\mathbf{p}) = \sum_{o \in \mathcal{S}} \lambda_o u_o(\mathbf{p})$$

2.5.4 Environment stability

Semantic terrain entities and environmental materials are inspired by the ecological concept of biogeocoenosis, which describes the relationship between living organisms (biotic) and their non-living environment (abiotic). These interactions closely mirror the processes observed in natural ecosystems, where biotic and abiotic components continuously influence each other, leading to a balanced and stable environment. Gubanov and Degermendzhy [CITATION] describe biogeocoenosis as almost-closed systems, with many parallels possible with thermodynamics such as the concept of dynamic equilibrium. In a biogeocoenosis, the various processes, such as the spread of nutrients, the growth of vegetation, or the erosion of soil, tend toward a state of balance.

Similarly, in our method, the interaction between semantic terrain entities and environmental materials is modeled using a reaction-diffusion-advection framework. This model captures how environmental materials are spread and absorbed within the terrain. Thanks to the decay rate of the environmental materials, the system progresses toward a dynamic equilibrium, where the distribution of environmental materials stabilizes. This stabilization reflects a state of balance where the rates of spreading and absorption and decay are equal, and the environmental attributes become coherent across the landscape. For example, a river might spread sediments downstream, which are gradually absorbed by the surrounding terrain, leading to a stable sediment distribution over time. Similarly, moisture emitted by a forest will eventually reach a balance with the surrounding soil and atmosphere, creating a stable, humid environment.

The only factor that can disrupt this equilibrium is a change in the environmental attributes. Changes such as an increase in temperature, a shift in wind patterns, or the introduction of a new semantic terrain entity can alter the balance, leading to a new phase of interaction and stabilization.

2.6 User interactions

The user can guide the generation process. The use of simple shapes as semantic terrain entities facilitate the edition of the simulation, as we can interactively add, remove or modify semantic terrain entities, or focus the generation process in a restricted area. Interaction with the environmental attributes is also provided as geomorphic events, that the user can invoke during the simulation. While the direct interactions on the semantic terrain entities are instantaneous, as a the geomorphic events are active on a given duration.

2.6.1 Direct interactions with the semantic terrain entities

The interactive nature of our simulation enables the user to modify the state of the terrain by manipulating directly the semantic terrain entities of the scene. We assume the modifications applied between two iterations of the simulation.

Translating an semantic terrain entity is trivial, we simply requires to evaluate the state of the semantic terrain entities at a translated position. The deformation of semantic terrain entities can be applied on curve and region semantic terrain entities by updating the control points of the skeleton and recomputing the resulting implicit surfaces. The evaluation positions used for region semantic terrain entities are displaced by applying a cage deformation of the 2D shape using the Green coordinates of points in the shape. After the alteration of the region, evaluation points should be keeping a similar distribution than before, avoiding unexpected results during the interaction. By modifying an semantic terrain entity, the environmental attributes may change, which can result in the destruction of the now incompatible environment objects in the scene (Figure 2.4).

As long as a non-zero fitness function is defined in the terrain, new semantic terrain entities can be forced by the user at any point of the simulation.

Control over the region of the terrain that should be updated can be given by adjusting all fitness functions through a scalar field $\lambda : \mathbb{R}^2 \mapsto \mathbb{R}$ such that the fitness function $\omega(\mathbf{p})$ of any new semantic terrain entity is evaluated as $\omega^*(\mathbf{p}) = \lambda\mathbf{p}\omega(\mathbf{p})$. This is especially useful in the planning of robotic simulations as we can first generate the overall shape of our terrain and secondly focus the generation process around the areas that may be visited by the robot, avoiding useless simulations and computer power. Figure 2.8 shows an example of colonization of the coral polyps that we limited manually into an annulus.

Our water current simulation is modeled as a simple vector field. As such, the user is able to interact with it at any moment of the simulation, allowing for the death of sensible semantic terrain entities while it will guide the simulation into a new landscape. By modifying the water currents, the user also modifies the transport rate of environmental materials at this position. The modification of currents is given as a stroke, a parametric curve C for which we evaluate $\Delta\mathcal{W}_{\text{user}}(\mathbf{p})$ just as for curved environment objects (Section 2.5.3).

2.6.2 Indirect interaction with semantic terrain entities

A configuration file can define in advance the different geomorphic events that should be triggered during the simulation. This can be useful to generate landscapes that are close to some existing locations. Multiple geomorphic events can be triggered either as sudden or continuous environmental changes. These changes play a huge role in the morphology of landscapes. We define geomor-

phic events with a starting point and an ending point, such that at any time of the simulation we can compute the progress of the geomorphic event as $t_e \in [0, 1]$.

Water level changes are important geomorphic events that shape the underwater landscapes. As previously submerged semantic terrain entities get elevated above water level, flora and fauna terrain features dry and die. Deprived from the living part of the features, everything is more affected by terrestrial erosion. By updating the value of the depth $\mathcal{D}$ evaluated in the fitness functions, any semantic terrain entity that is sensible to the depth will be impacted automatically, that may be causing death (Figure 2.5). The modification of the water level is defined as

$$\mathcal{D}(\mathbf{p}) = \mathcal{D}_0(\mathbf{p}) + \sum_{e \in \text{events}} \Delta\mathcal{D}_e t_e$$

with $\Delta\mathcal{D}_e$ the amount of water rising or lowering during an geomorphic event. We assumed a linear evolution of the water level during an geomorphic event. This allows to evaluate the depth at any point in space and in time.

Subsidence and uplift are the main geomorphic events that create or destroy islands in the long term. These geomorphic events are simulated as a simple factor on the height field of the generated terrain (Figure 2.6). Subsidence is not always uniform in the terrain. As such, the user can provide a position $\mathbf{q}$ at which the subsidence is the strongest, the amount of subsidence applied $\Delta\mathcal{H}_e$ and a standard deviation σ for which we can then compute at any point in space and time of the simulation the height of the terrain

$$\mathcal{H}(\mathbf{p}) = \mathcal{H}_0(\mathbf{p}) \cdot \sum_{e \in \text{events}} \frac{G(\|\mathbf{p} - \mathbf{q}\|)}{G(0)} \Delta\mathcal{H}_e t_e$$

with $G(x)$ the Gaussian function

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

Storms are factors of the geomorphology of coral reefs (Oron et al., 2023; Vila-Concejo and Kench, 2016) and coasts (Cowart et al., 2010; Domínguez et al., 2005). Due to the extreme wind and wave velocities coasts are highly eroded in a short time period and the more fragile corals near the water surface are broken, possibly causing breaches in the reefs and spreading polyps in the currents direction. While there are many factors at play to understand the apparition of storms and the hydrodynamics affecting it, we simplified the model of storms to the user as a single epicenter $\mathbf{q}$ with a wind velocity v_{wind} and a standard deviation σ representing the spread around the epicenter (Figure 2.7). The

computation of water currents are then computed as

$$\mathcal{W}_{\text{user}}(\mathbf{p}) = \mathcal{W}^*_{\text{user}}(\mathbf{p}) + \sum_{e \in \text{events} | t_e \in [0,1]} v_{\text{wind}} \frac{G(\|\mathbf{p} - \mathbf{q}\|}{G(0)}$$

In this case, we did not include the linear factor t_e as storms are usually conserving a constant force for the time of the few weeks or months of their occurrence.

with ΔT_e the change of heat during an geomorphic event, T_0 the temperature at the water surface, and c a very small factor.

The framework can easily be extended as the geomorphic event system stays similar for all geomorphic events. Including higher level simulations in the geomorphic event system can be added, such as the simulation of tectonic activity, the use of fluid dynamics for tsunami geomorphic events, the integration of human activity, ...

2.7 Results and discussion

Our method provides a way to generate scenes at different scales. We demonstrate this capacity with the generation of a large scene of an island (Figure 2.1) after what we focused the generation process in a canyon (Figure 2.9), then a small-scale visualization of coral colonies (Figure 2.8). In the examples, we rendered the semantic terrain entities as a implicit tree or as individual meshes. The island, lagoons, reefs, canyons and sand ripples as implicit surfaces

A canyon scene can be generated using our method. The water flow is affected by the curve of the canyon such that the currents are oriented in the direction of the curve's tangent.In this example, we force the position of arches to be inside the canyon. The arches deposits a material "rock deposit", which is the main element of the fitness function of the Rock object. The "rock deposit" is slightly affected by water currents, but its mass make it highly affected by gravity. As such, rocks will spawn underneath arches. In reality, an arch is often created as part of a large coral boulder that sees the calcareous bottom part detached by the water currents, often resulting in an arch surrounded by big rocks and smaller rocks from the erosion of the first rocks. As such, we define an semantic terrain entity "Arch" with a fitness function $\omega_{arch}(\mathbf{p}) = 5 - d(canyon - \mathbf{p}) * \|\mathcal{W}(\mathbf{p})\|$, an semantic terrain entity "Rock" using $\omega_{rock}(\mathbf{p}) = \mathcal{M}_{rock\_deposit}(\mathbf{p})$ and Pebble using $\omega_{pebble}(\mathbf{p}) = \mathcal{M}_{smaller\_rock\_deposit}(\mathbf{p})$. Finally, sand ripples are simply described as curves appearing where there is a lot of sand available: $\omega_{ripple}(\mathbf{p}) = \mathcal{M}_{sand}(\mathbf{p})$. Following these simple rules, Figure 2.9 shows the emergence of details in the scene.

In this example we defined three different types of corals, coralA, coralB and coralC, to illustrate the possibility to model behaviours from the choice of fitness functions. Each of the coral types deposits a material "coral polyp" and "coral polyp A" ("coral polyp B" and "coral polyp C" respectively). By considering a fitness function that minimize the ratio $\frac{\text{coral polyp}}{\text{coral polyp A}}$, we can see an emergent behavior of the three types of coral fighting for the space colonization. Figure 2.8 shows the result of this simulation at three different interations. At the border between two colonies, none of the colonies make progression due to the amount of coral polyp specific from the other colony.

The proposed method aims to generate plausible landscapes using simplified versions of the evolution of an ecosystem and of the 3D representation. The biological realism of the result is highly correlated to the amount of simplification and assumptions, while the visual realism is completely dependent to the geometric functions used for the 3D modeling of the semantic terrain entities. While proposing a flexible method that propose a generic approach for terrain generation, a close collaboration with fields experts and with graphists is needed to achieve optimal results.

Most simulation algorithm's quality depends on the size of the time step used, but with the introduction of a decay rate in the environmental materials properties, we limit the influence of time steps by considering that steady-state are reachable. The material deposition and absorption on punctual semantic terrain entities can be seen as a Dirac function δ centered at their position resulting in the advantage that material displacement function can use the definition of the diffusion equation instead of the advection-diffusion-reaction equation. This equation allowing us to evaluate the state of the material $\mathcal{M}$ without intermediate steps, but this is not applicable with curve- and region-based semantic terrain entities.

2.8 Conclusion

We have proposed a method to generate terrains procedurally using sparse representations. This representation, the semantic terrain entities, enables to introduce expert knowledge by the mean of the fitness functions that rule the semantic terrain entities life cycle, but also to integrate the user in the loop during the generation process. We reduced the terrain resolution limitations by defining the environment objects as parametric features. Thanks to the sparse representation based on single points, curves and regions, we allow for direct manipulation of the semantic terrain entities of the scene by the user which, thanks to the environment steady state consideration, also enables to include

these interactions in the automatic simulation process. Integrating environmental properties in the fitness function of semantic terrain entities allows the user to guide the generation through geomorphic events. Our method enables each semantic terrain entity of the scene to influence the environment locally, reducing the need of computations while also retrieving environmental attributes locally, which result in a parallelizable life-like simulation process. The genericity of the environment properties definitions should be sufficient for plausible generation of other landscape types as long as expert knowledge can be translated to semantic terrain entity's formalism.

We limited our work to the use of 2D scalar fields as they are more easily differentiable, interpretable and lighter than volumetric representations. However, future works include using 3D representations of the terrain and the environment to generate 3D terrains, including cavities, sub-terrestrial areas and the interior of coral structures.

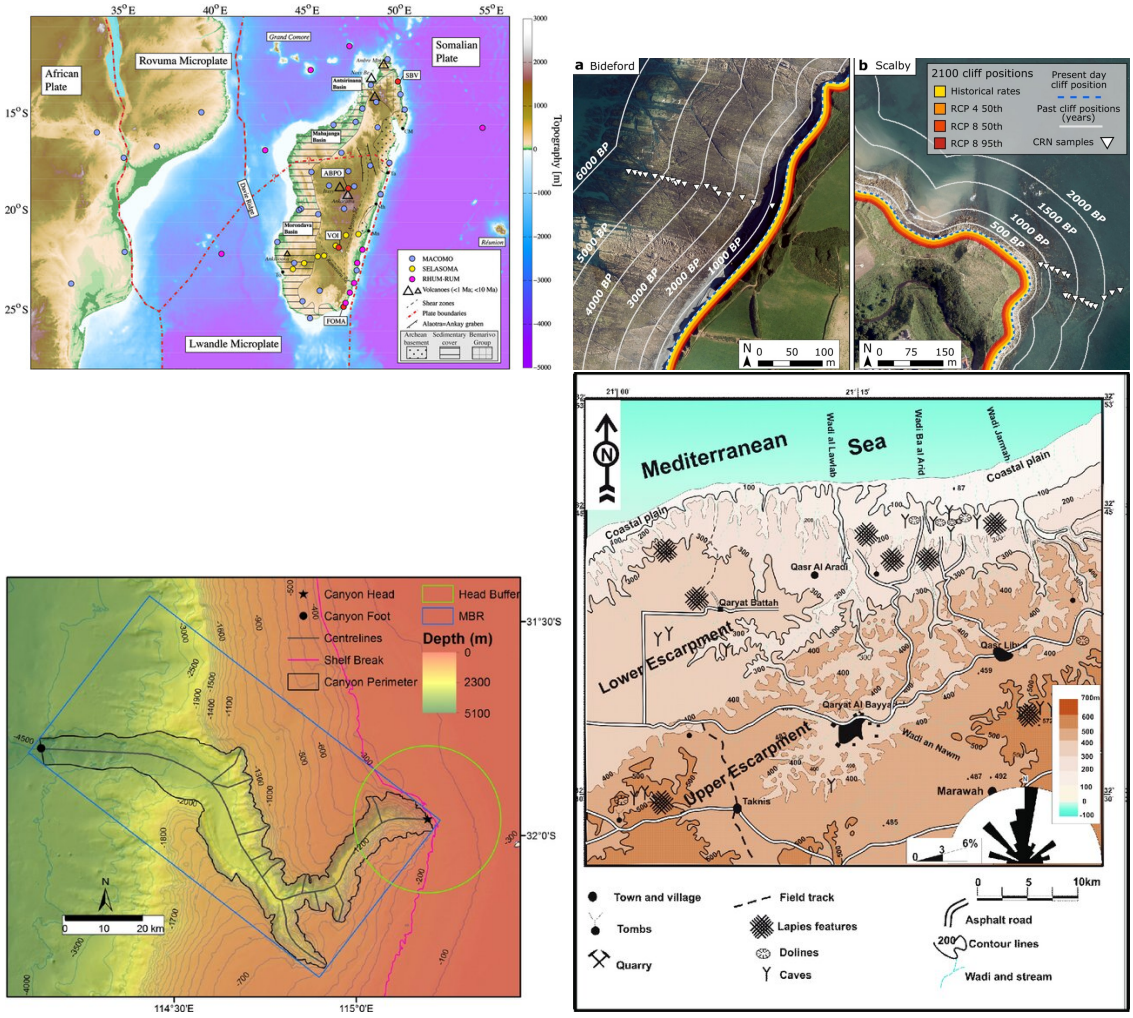


Figure 2.2: Four examples of topographic maps used in earth science. From left to right and top to bottom: Sedimentary distribution other Madagascar island (Pratt et al., 2017), evolution of coastlines at Bideford, UK and Scalby, UK over the last 6000 years and 2000 years respectively (BP = Before Present) (Shadrick et al., 2022), localization of key parameters of Perth Submarine Canyon, Autralia (Huang et al., 2014), and geological features distribution of karstic landscape at Qasr Lybia, Libya (Amawy and Muftah, 2009)

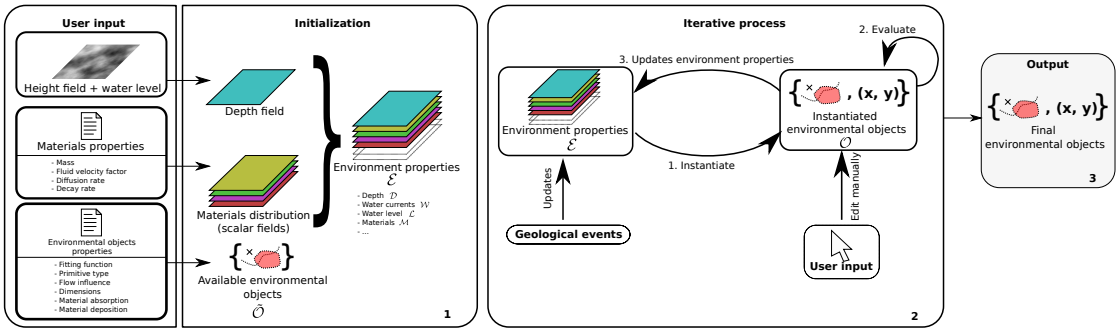


Figure 2.3: Overview of the pipeline of the method. The user provides as input an initial height field and sets the water level, as well as a definition of the environmental materials properties and semantic terrain entities properties that will be used in the iterative process. These inputs are initialized as an initial set of semantic terrain entities and scalar fields that represents the environmental attributes. In the iterative loop, new semantic terrain entities are instantiated using the current state of the environment at their optimal position. The existing semantic terrain entities in the terrain reevaluate their fitness function to grow or die and update the environmental attributes locally. At each iteration, geomorphic events can update the environmental attributes, while the user can interact directly with the semantic terrain entities. The result of the whole process is a set of semantic terrain entities which is a sparse representation of the features of the scene.

Figure 2.4: Starting from a coral colony developed around a canyon (*left*), the user edits the shape of the canyon, resulting in a different configuration of the scene, killing the corals that ends too deep in the water (*center*) and the development and growth of new corals at the previous location of the canyon (*right*).

Figure 2.5: Lowering the water level by a few meters caused most of the coral objects to satisfy $\omega \leq 0$, causing their death. Since the water level (blue) decrease slowly, new coral objects spawn progressively at a lower altitude.

Figure 2.6: Simulating subsidence on a part of the terrain (brown area) cause the depth value to change locally, resulting in the death of coral objects that find themselves too deep to survive. Here two subsidence geomorphic events are triggered in parallel.

Figure 2.7: The result of a storm localized on one side of the island (red area) modifies the result of the evaluation of semantic terrain entities around its epicenter for a short period of time. Most of the coral objects died from the geomorphic event, except few semantic terrain entities less sensible to water currents strength.

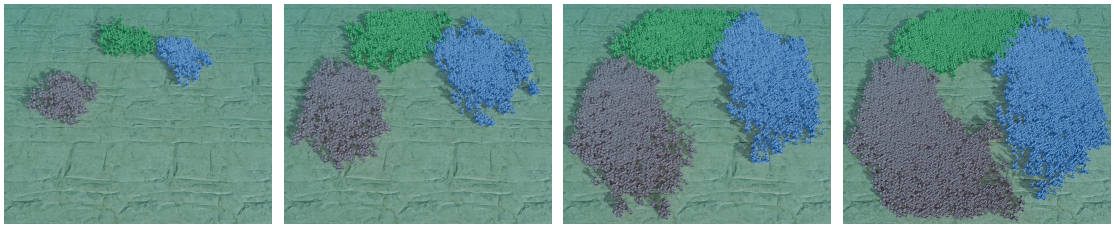


Figure 2.8: Three colonies of coral (red, blue, green) restricted to an annulus the middle section of the terrain fighting for the space.

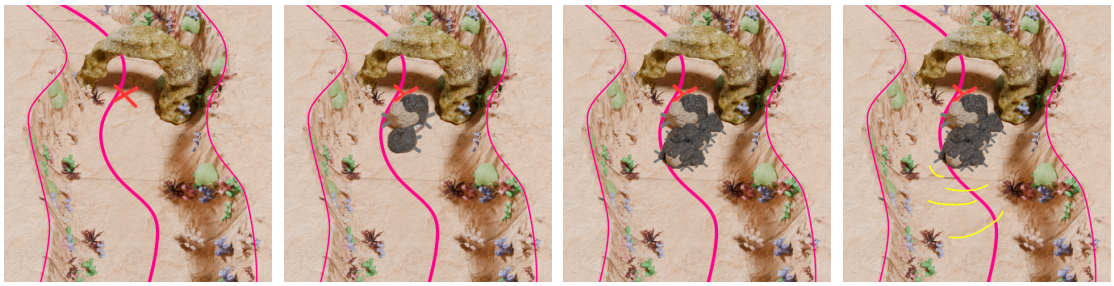


Figure 2.9: Evolution of a canyon scene at different iterations of the simulation. The apparition of an arch causes the spawning of rocks, pebbles, and finally some deposition of sand at the bottom of the canyon, spawning ripples.

Figure 2.10: A simple coral island is generated using an island, a lagoon, reefs coral polyps, beaches, trees and algae semantic terrain entities. Trees appear on beaches and algae grow in the lagoon's sand.

Part II

Modelisation

Abstract

- Methods completely different
** Physical phenomena are different
** => Simulation/generation methods must be specific for one landscape
- Methods are developed at different instants of the thesis work,
** We will see different procedural generation domains:
*** Analytical solutions (coral islands #1)
*** Deep Neural Networks (coral islands #2)
*** 3D user interaction (karst networks)
- Deep Learning tends to replace procedural methods in 2D domain,
** But still too complex for 3D models
*** (lack of data, lack of research interest for now)
** Still requires a lot of data, which, is (while not easy) possible using aerial images, but is way too sparse with underwater landscapes or underground biomes.
- We want to control the area in which an element is modeled in the terrain
** Because that's how we define them in the previous chapter.
** Thus we cannot use simple random noise methods => no bounds
*** Only solution is to use falloff maps, but meh...
- As such, we want to keep the skeleton of our elements using primitives (points, curves, regions)
** => Much easier to add constraints / manipulate primitives in a "procedural generation" way.
- Warning: usage of Deep Learning is at the limit of "procedural generation" and is not considered as part of it by the whole community.
(Complete with the Reddit poll).
- ...

CHAPTER 3

Graphical representation of environmental objects

- Implicit surfaces
- Meshes
- ...

CHAPTER 4

Automatic Generation of Coral Islands

Contents

Back to summary

4.1 Introduction

- Definition of coral islands
** Different types of coral islands

*** Here, volcanic islands
- Presentation of corals
- Difference from regular landscapes
** Concept of corals
*** Long-term evolution (island) and short-term evolution (corals)
**** Geological process of island subsidence
- ...

4.1.1 Multiple theories

- Complexity of coral reef ecosystems
** Biological diversity
*** Varied species of corals and their differing growth patterns
*** Interaction with marine flora and fauna
** Environmental factors
*** Influence of water temperature, salinity, and light availability
*** Impact of ocean currents and wave action
- Geological processes
** Dynamic nature of earth's crust
*** Plate tectonics and volcanic activity
*** Subsidence and uplift processes
** Variations in sea levels
*** Historical fluctuations due to glacial and interglacial periods
*** Current sea level rise and its impact on coral reefs
- Historical and technological context
** Historical developments in marine science
*** Early exploration and observations by naturalists like Darwin and Wallace
*** Advances in geological and oceanographic methods
** Technological advancements
*** Development of deep-sea exploration tools
*** Use of sonar mapping, core sampling, and seismic surveys
- Limitations of early theories
** Inadequate data and observation
*** Limited access to deep-sea environments in the 19th and early 20th centuries
*** Reliance on surface observations and anecdotal evidence
** Evolving scientific understanding
*** Changes in scientific paradigms and theories over time
*** Integration of new findings and methodologies
- Different perspectives and disciplines
** Geological vs. biological perspectives

*** Focus on geological processes like subsidence and sea level change
*** Emphasis on biological factors such as coral growth and reproduction
** Interdisciplinary approaches
*** Collaboration between geologists, marine biologists, and oceanographers
*** Diverse methodologies leading to different interpretations
- Regional and case-specific variations
** Geographical differences
*** Variations in reef types and formations across different regions
*** Influence of local environmental conditions and geological settings
** Case studies of specific islands
*** Unique formation histories of individual atolls and coral islands
*** Examples from the Pacific, Indian, and Atlantic Oceans
- Ongoing research and discoveries
** New findings and theories
*** Continuous exploration leading to new hypotheses and models
*** Advances in climate science impacting understanding of historical sea levels
** Reevaluation of existing theories
*** Critical assessment of long-standing theories with new data
*** Adaptation and refinement of theories over time

Theory 1: Subsidence theory

- Origin and proponents
** Charles Darwin (The Structure and Distribution of Coral Reefs, 1842)
** Alfred Russel Wallace
- Mechanism
** Initial formation around a volcanic island (fringing reef)
** Gradual sinking of the volcanic island (subsidence)
** Transition to a barrier reef with a lagoon
** Complete submersion of the volcanic island leading to atoll formation
- Supporting evidence
** Observations from the HMS Beagle voyage
** Modern geological surveys and core samples

Theory 2: Growth on submarine mountains theory

- Origin and proponents
** John Murray
** Alexander Agassiz
- Mechanism
** Coral colonization on underwater mountains (guyots)

** Vertical growth of corals towards the sea surface
** Formation of atolls without the need for subsidence
- Supporting evidence
** Studies on deep-sea coral formations
** Distribution of guyots and seamounts in tropical regions

Theory 3: Sea level change theory

- Origin and proponents
** Reginald Daly (The Coral Reef Problem, 1915)
- Mechanism
** Impact of glacial and interglacial periods on sea levels
** Lowered sea levels exposing coral reefs, allowing vertical growth
** Rising sea levels creating conditions for atoll formation
- Supporting evidence
** Geological records of sea level changes
** Correlation with coral reef growth periods

Theory 4: Erosion and sedimentation theory

- Origin and proponents
** Maurice Ewing
** William Donn
- Mechanism
** Erosion of coral reefs by waves and currents
** Accumulation of coral debris forming islands
** Continuous process of erosion and sediment deposition
- Supporting evidence
** Sediment analysis around coral reefs
** Observations of island formation from coral rubble

Theory 5: Platform reef theory

- Origin and proponents
** William Morris Davis
- Mechanism
** Coral growth on stable continental or insular platforms
** Formation of reef structures (platform reefs)
** Development into coral islands when conditions are favorable
- Supporting evidence

** Studies on platform reefs and their distribution
** Geological stability analysis of coral platforms

Comparative analysis

- Similarities and differences
** Common themes: coral growth, geological activity, sea level changes
** Differences in primary mechanisms (subsidence vs. growth on seamounts)
- Complementary aspects
** Integration of multiple theories for a comprehensive understanding
** Case studies showing multiple processes at work

Modern advances and technologies

- Geological surveys
** Core sampling techniques
** Seismic and sonar mapping
- Environmental studies
** Impact of climate change on coral growth and sea levels
** Conservation efforts and their implications for island formation

4.1.2 Darwinian theory

- Several theories
- Difficulty in studying environments
** Use of observations
- Theory refuted by (Droxler and Jorry, 2021)
** But it's too early to judge
** Practical for our case.
- ...

4.1.3 Overview

- Proposed tool
** Sketching the island
** Sketching the profile
** Wind simulation
- Automation of examples
** Data augmentation
- ...

4.2 Related works

- Perlin + falloff
** But lacks control
- Uplift [SCHOTT UPLIFT] (Cordonnier et al., 2016; 2017a)
** But we propose a method that limits the required interaction
- Sketching [PEYTAVIE SKETCHING, EMILIEN FIRST PERSON] (Gain et al., 2009)
** Added radial constraint to simplify interaction
- Geological modeling (Patel et al., 2021)
** For us, it's hard to know what's happening underground (?)
- Unlike the literature, integration of the underwater part
** Integration of observation data into our process
*** -> Typical shape and profile of an island
** Difficult to integrate physical simulations due to uncertainty
- Layer-based generation could improve the realism of examples by considering layer/environment interactions.
- ...

4.3 Example generation

- We use Darwin's theory in our case because generation is relatively simple.
- 2 steps:
** Generation of the island
** Generation of the reef
- Numerous assumptions:
** Island has a relatively round shape
** Coral grows to a constant height around the island
** All "features" are radial
** Deformations of islands caused by winds and waves
** Islands are independent of each other
** The profile is relatively identical all around the island
- ...

Input

- Sketching of the island from above + profile view
- Wind strength
- Water level

- Subsidence rate
- ...

"Simulation"

- Subsidence calculated by simple scaling
** Note: No geological consistency
*** Here, using a zero level to scale in Z
*** No consideration of different soil materials
- Reef growth
** Does not consider any weather events
** Considers the "keep up" strategy (as opposed to "give up" and "catch up") [large simplification]
- Wind deformation
** using directly $f(\mathbf{p}) = f(\Phi(\mathbf{p}))$.
- ...

Output

- Height map of the island's surface
- Island zones and coral zones
- Possibility to recalculate ground height and coral height
- ...

4.3.1 Closed form of coral growth

- Proposes a closed-form solution for surface calculation
- Calculation of ground height:
** Calculation of height by revolution around the origin point
** Deformation of the revolution profile using the top-down sketch
** Deformation of the height field by the wind map.
- Calculation of growth:
** On the initial heightmap,
*** Any surface $z_{min} < h(p) < z_{max}$ becomes coral
*** Calculation of the "low" contour $h(p) = z_{min}$ and "high" contour $h(p) = z_{max}$
*** ...
- Deformation of the map using the wind map:
** ...
- ...

4.3.2 Labeling of the map

- Using top-down sketch:
** Features "Mountain", "Island borders", "Beach" are radial (θ, r), so we can label each point of the map as the "next" feature
- Using coral simulation information:
** Provides the labels "Lagoon" and "Reef {begin, peak, end}". - The height map is directly associated to the feature map
- This is perfect to feed a cGAN.
- ...

4.3.3 Automation

- Take advantage of the radial nature of the features
- Some features can be optional (mountains)
- Deformation of the feature lines
** Influence on the radius: $\tilde{r}(\theta) = r(\theta) + \eta(\theta)$ with η continuous noise function 2π-periodic.
- ...

4.4 cGAN

- Conditional GAN: a type of Generative Adversarial Network (GAN) where the generation process is conditioned on additional information.
- ...

4.4.1 Definition of cGAN

- Two networks: consists of two neural networks, a generator and a discriminator, which compete against each other.
** Generator: takes both random noise and additional information (like class labels or data) to produce synthetic data.
** Discriminator: evaluates whether a given data instance is real (from the actual dataset) or fake (produced by the generator), while also considering the additional information.
** Additional information: this can be labels, data from other modalities, or any other contextual information that guides the generation process.
- Training process: the generator tries to create realistic data to fool the discriminator, while the discriminator tries to correctly classify data as real or fake based on both the data and additional information.

- Objective: the goal is to improve the generator's ability to produce realistic data that matches the given conditions and to enhance the discriminator's ability to distinguish between real and fake data.
- Applications: used in various domains including image-to-image translation, text-to-image synthesis, and other tasks where generating data based on specific conditions is required.
- ...

4.4.2 Why cGAN?

- Flexibility of input
- Moving beyond the "radial" input condition
- Output even for "inconsistent" data (e.g., ocean in an island)
- No math, geometry, geology, or complicated things to master (hehe)
- ...

4.4.3 Training

- ...

Use of synthetic data

+ Problem with synthetic data
- ...

Data augmentation

- ...

4.4.4 Model usage

- ...

Generation from sketch

- ...

Interactive times

- ...

Realism

- ...

CHAPTER 5

Generation of karst networks

Contents

Back to summary

- ...

5.1 Introduction

- Definition: complex subterranean systems formed primarily in soluble rocks like limestone, dolomite, and gypsum.
- Importance: significant for water resources, unique ecosystems, and land use management.
- Formation and development
** Chemical weathering
*** Study the role of rainwater acidity (carbonic acid) in dissolving carbonate

minerals.
** Underground drainage
*** Analyze the development of subterranean drainage systems, including caves, tunnels, and sinkholes.
** Speleogenesis
*** Examine the process of cave formation, especially at or below the water table.
- Key features of karst networks
** Caves and caverns
*** Document major cave systems and their formation processes.
** Sinkholes (dolines)
*** Identify areas prone to sinkhole formation and study their causes.
** Disappearing streams and springs
*** Trace the path of streams that vanish into the ground and re-emerge.
** Karst valleys
*** Investigate valleys formed by the collapse of underground voids.
** Stalactites and stalagmites
*** Study the formation of these features within caves.
- Hydrology of karst networks
** Aquifers
*** Assess the productivity and structure of karst aquifers.
** Groundwater flow
*** Model the rapid and turbulent flow of water through karst systems.
** Vulnerability to contamination
*** Evaluate the risks and sources of contamination in karst aquifers.
- Ecology of karst environments
** Unique habitats
*** Research cave-dwelling species (troglobites) and their adaptations.
** Biodiversity hotspots
*** Identify and document biodiversity hotspots in karst regions.
- Human interaction with karst networks
** Water resources
*** Study the dependence of regions on karst aquifers for freshwater.
** Tourism
*** Assess the impact of tourism on karst landscapes and caves.
** Land use challenges
*** Develop guidelines for construction and land use planning in karst regions.
** Environmental concerns
*** Identify and mitigate pollution and land degradation impacts.
- Examples of karst regions
** The Mammoth Cave System (USA): world's longest cave system.

** The Guilin Karst (China): unique limestone peaks and river systems.
** The Dinaric Karst (Balkans): extensive cave systems and karst phenomena.
- Research and study in karst science
** Speleology
*** Promote the scientific study of caves and karst features.
** Geohydrology
*** Conduct studies on water flow in karst systems for resource management.
** Geomorphology
*** Research the development of karst landforms over geological timescales.
- Action steps
** Field surveys and mapping
*** Conduct detailed field surveys and create maps of karst features.
** Hydrological studies
*** Implement hydrological modeling and water quality testing.
** Ecological research
*** Carry out biodiversity assessments and ecological studies in karst habitats.
** Human impact analysis
*** Study the impact of human activities on karst systems and develop mitigation strategies.
** Education and outreach
*** Educate local communities and stakeholders about the importance of karst networks and sustainable practices.
- Goals
** Conservation
*** Preserve unique karst ecosystems and landscapes.
** Sustainable development
*** Ensure that human activities in karst regions are sustainable and minimize environmental impacts.
** Scientific advancement
*** Promote research and understanding of karst processes and features.

5.2 Related works

- State of the art:
** (Paris et al., 2021)
*** In our case, we do not rely on a graph and path finder, which allows us to compute sections of the karst on the fly (no need to know the whole network to find paths)
** (Pytel and Mann, 2015)
*** Based on voxels, looks plausible, with a low number of parameters, but take

way too long (~10 to 20 minutes per generation)
*** Which makes it unfit for user interaction
** (Collon et al., 2015; 2017)
- With some imagination, we can see the shape of trees:
** [Prusinkiewicz : ANY]
** (Runions, 2008)
- Or miscelleanous networks:
** (Fernandes and Fernandes, 2018; É. Galin et al., 2010)
- Maybe even look for lychen or ant colonies
- ...

5.3 Space colonization

- Not really a tree, but...
** Some networks are branching
** Some have low number of cycles
- SC is easy to manipulate for an user
- For these reasons, we will consider the use of SC.
- Description of the algorithm:
** Definition of a root and sinks
** Evolution from the root toward closest sinks
** Branching [WHEN NEEDED]
- ...

5.4 Our method

- Based on classic SC
** Merging branches
** Closing paths based on angle and distances to create cycles
- Adding width to the branches
** Destroying paths when width too small
- Leaves as chambers/cavities
- ...

5.5 Modeling

- Tunnels:
** The output of SC is a set of paths

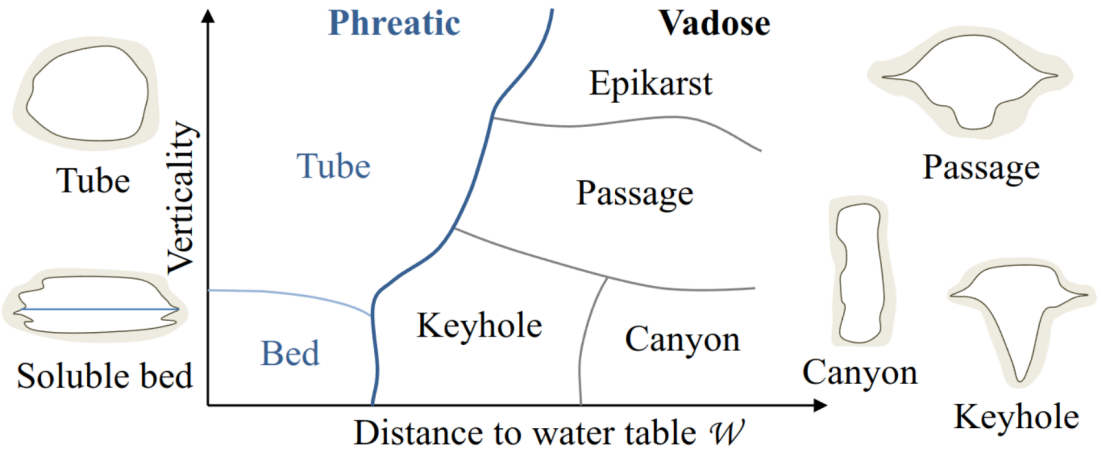


Figure 5.1: Classification of tunnel shapes

** (Paris et al., 2021) provides a classification of tunnel shapes (5.1).
- ...

5.6 User control

- Importance of keeping user control
** Shape of a karst is close to randomness
** Want to be predictible
- Manipulation of control points
** Source
** Sink
- Paths tortuosity
- Inclusion of soil properties in the formation of paths, using the gradient of:
** Humidity,
** Porosity
- Real-time editing
** Allows for precise manipulation
- ...

5.7 Results

- ...

5.8 Conclusion

- ...

Part III

Erosion Simulation

Abstract

- Work carried out at the beginning of the thesis
- Representation choice still uncertain
- Search for abstraction of the representation
** Drives the generalization of the erosion system

CHAPTER 6

Érosion par particules

Contents

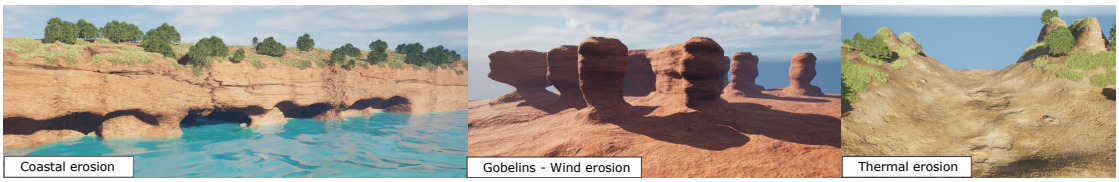


Figure 6.1: Applying shading and textures on the generated geometry can produce a plausible aspect of a coast eroded by waves on a long timespan, or a desertic landscape eroded by wind, or a mountainous area flatten by thermal erosion.

Back to summary

6.1 Introduction

Automated terrain generation is a key component of natural scene digital modeling for animated movies and video games. A standard approach is to first generate a base terrain geometry using noise to define the height on the input domain (Musgrave et al., 1989; Olsen, 2004; Roudier, 1993), the result will most likely lack realism and feel synthetic. Erosion simulation algorithms are applied, to simulate thousands of years of ageing by reproducing physical phenomena - i.e. effects of the elements (rain, wind, running water...) - affecting the terrain making it more believable (E. Galin et al., 2019; Smelik et al., 2009; Stachniak and Stuerzlinger, 2005).
The process of terrain alteration caused by the effect of water, air, or any other element - natural or not - over time is usually performed in three steps (Neidhold et al., 2005): **detachment** - pieces of the ground of variable dimensions, ranging from complete ledges to grains of sand, are removed from the terrain depending

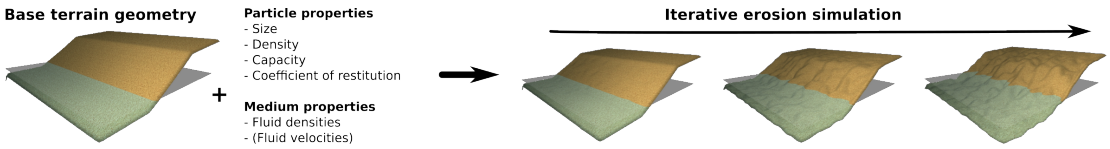


Figure 6.2: Our method require a base geometry, a small number of parameters for the particles and the medium used for the erosion simulation. It can be easily adapted to be compatible with different mediums and terrain representations.

on the simulated meteorological phenomenon - **transport** - pieces of ground fallen from their initial position are moved to a different one (e.g. a cornice falls down a slope or a grain of sand is thrown into the air) - and **deposition** - transported pieces of land are accumulated at a new part of the landscape. Various phenomena can cause these alterations: **thermal erosion** (bursting of rocks caused by expansion of water under frost, then falling of debris to the bottom of a slope), **hydraulic erosion** (detachment caused by the impact of water particles on surfaces and the transport of sediments by the flow of runoff), **wind erosion** (fine particles carried away in the wind and hit surfaces on their way, creating new fine particles which then also fly away), **chemical erosion** (chemical decomposition of rocks caused by rainwater or other fluids), other exceptional phenomena such as avalanches, animals, lightning, etc... modify the terrain (Argudo et al., 2020; Cordonnier et al., 2017b; 2017a; 2018; 2023).

In practice, the core idea to simulate erosion is to add or remove material from the terrain at given positions on the interface between the terrain and fluid eroding it (e.g. air or water). Hence, the two major problems to tackle are: how to locally alter the terrain geometry for material detachment and deposition and where to perform these alteration given the properties of the environment (terrain slope, fluid density and velocity). A terrain is more than often represented in 2.5D using a 2D image called a heightmap which grey scale values define terrain elevation. While being the major terrain representation, only a limited number of environments can be modeled. Indeed, natural landscapes are intrinsically 3D (overhangs, cavities or geological structures such as arches or gobelins), this is particularly true for underwater environments generation. Alternate representation such as voxel grids, material layers or implicit surfaces can be used. A wide variety of method have been proposed to simulate natural erosion phenomena on heightmaps as the partial differential equations to model erosion can be discretized and solved in 2D and the material detachment and deposition at a given point of the terrain surface can be easily performed by elevating or lowering the ground level i.e. changing locally pixel intensities. For volumetric representations, the alteration of the terrain is not as trivial. To define where to

perform the erosion process the local slope variations are more than often used combined with eroding medium information. This fluid can be simulated using particle systems, Smoothed Particle Hydrodynamics (SPH) (Krištof et al., 2009) or approximated using a simple vector field. Proposed methods offer a specific erosion effect tailored to a single terrain representation and fluid simulation.

In this work we propose an approach to simulate a large part of the geomorphological and meteorological phenomena present in the literature of terrain generation (including 3D and volumetric effects). We introduce a generalized algorithm performing the three stages of erosion on surface and volume representations alike, and expose very few intuitive parameters to be adjusted by the user (Figure 6.2). We propose to tackle separately the material variation and the fluid simulation. Our method relies on a particle system to simulate eroding agents, each thrown particle will collide with the terrain, perform terrain alteration at the collision point and transport material along its path. Their motion is computed using simple particle physics accounting for the medium density and particle properties (buoyancy and gravity forces). We consider each particle as independent, hence, they do not interact with each other, no collision detection or response. This simplification allows for efficient parallel computation. When more accuracy or control is needed, we propose to provide a vector field used to modify the particle speed at each time step. The nature of this vector field is flexible, it can be computed using a more or less accurate fluid simulation (SPH, FLIP,...) or be manually defined by the user. We propose a particle-based strategy for material alteration that can be applied on surface and volumetric representation.
The main contributions of this paper are:

- a generalized particle-based algorithm performing the three stages of erosion on surface and volume representations,
- decoupling the erosion system from the fluid simulation, making the process more flexible in its usage and implementation and opening the door for richer effects that can easily be produced.

6.2 State of the art

In this section, we first present the major terrain representations (height fields, layered representations, voxel grids, and scalar functions) and a subset of the major simulated phenomena used to erode terrains. We highlight the fact that, in the literature, a specific erosion method tailored to a given terrain representation is proposed for given phenomena which might lead to limitation

in term of terrain modeling. Indeed, changing representation costs information and precision loss.

6.2.1 Terrain representations

A terrain can be represented in various ways, each of them suited for a given application of which we give an brief overview, more details can be found in (E. Galin et al., 2019).

Height Fields

Height fields represents the surface of the terrain by defining the elevation at each point in a 2D grid. This representation is simple, regular, and fast to process allowing for easy manipulation, such as raising or lowering the terrain (Emilien et al., 2015; Gain et al., 2009).

Layered Representations

Layered representations are an extension of height fields using a 2D grid where each cell represents a stack of different materials instead of a simple height (Beneš and Forsbach, 2001; Peytavie et al., 2009) allowing for memory efficient representation of volumetric terrains. To create complex structures, arches or caves, solid materials can be transformed into more granular ones, that can be stabilized (Peytavie et al., 2009).

Voxel Grids

Voxel grids are regular, uniform volumetric grids that encode information on the presence or absence of material for each 3D point in the domain. Voxel grids are advantageous due to their regularity (Dey et al., 2018) and ability to represent volumetric models at the cost of high memory footprint, which has limited their use in terrain generation (Ito et al., 2003; Kaufman et al., 1993; Lengyel, 2010). We consider two voxel grid representations : *density-voxel* grids for which each voxel contains a scalar value, for instance the occupation percentage (Eisemann and Decoret, 2008) and *binary voxel* grids that can be seen as a mask containing the presence of material information.

Implicit terrains

Implicit terrains represent landscapes as an implicit surface defined by a scalar function. This allows for high definition large terrain modeling. The

application of combinable scalar function overlays (É. Guérin et al., 2016) or the definition of user-defined gradients (E. Guérin et al., 2022) can be used to create complex terrain features. Altering a implicitly defined surface is a challenging task hence limited option exist for erosion simulation (Paris et al., 2019b).

6.2.2 Erosion processes

Erosion processes play a crucial role in shaping landscapes over time. We present different kind of erosion and how they apply to given terrain representations. Note that using existing methods all erosion methods can not be used on all representation.

Thermal erosion

Thermal erosion is driven by large temperature shifts, transferring material based on slope thresholds. The process is iterative, redistributing material until slopes stabilize. It can be computed efficiently on height fields and layered terrains due to their manipulable height nature (Beneš and Forsbach, 2001; Musgrave et al., 1989; Peytavie et al., 2009). However, its application on voxel grids is challenging due to limited Z-axis resolution.

Hydraulic erosion

Hydraulic erosion stems from water movement, eroding and depositing sediment based on water flow intensity. 2.5D terrains are widely studied for this simulation, using either water slope velocities (Neidhold et al., 2005) or water simulations for erosion effects (Mei et al., 2007). For smaller scales, 3D fluid simulations on voxel grids have been proposed (Beneš et al., 2006). Kristof et al (Krištof et al., 2009) used SPH (Smoothed Particle Hydrodynamics) for meshless erosion simulations on various terrains. Their method involves numerous particle interactions, demanding significant computational power. Our approach draws inspiration from this but enhances efficiency by removing certain particle interactions.

Wind erosion

Wind erosion shifts material through wind force, notably impacting areas with fine surface particles like deserts. It has been modeled on discrete height fields (Paris et al., 2019a; Roa and Benes, 2004) by mimicking sand's wind-driven trajectory and using thermal erosion for corrections. This process is simulated

by iteratively displacing small amounts of matter, which make it less suitable for representations with discrete height resolution.

Erosion by other forces

Erosion comes in many forms. These includes influences like glaciers, snow, tectonic movements, and fauna, each introducing distinctive terrain patterns, enriching its intricacy (Argudo et al., 2020; Cordonnier et al., 2016; 2017b; 2017a; 2018). However, most methods are tailored by a given terrain representation, often the height fields, and might not be applicable to other representations due to their intrinsic properties.

6.3 Particle erosion

Erosion occurs in three stages: material detachment, transport and deposition (respectively in red, black and green in Figure 6.3). In our approach, particles move through the medium following its flow (i.e. wind in air or currents in water) and then absorb or deposit a small amount of material upon contact with the land surface, effectively fulfilling the three stages of erosion.

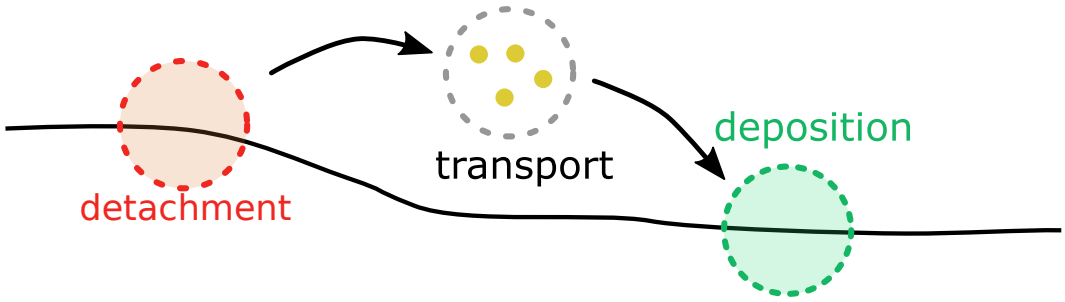


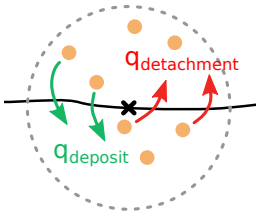
Figure 6.3: Three steps of the erosion process from the sediment point of view: detachment from its original location - dotted red circle -, transport in a fluid - dotted black circle -, deposition at a new location - dotted green circle.

6.3.1 Overview

Particles are transported through the medium and can pass through several different media. Each medium is defined by a density and a flow. Consider, for example, water density to be $1000\,\mathrm{kg\,m^{-3}}$ and that of air to be $1\,\mathrm{kg\,m^{-3}}$. The gravity applied to the particles is then very different between open and submerged environments due to the difference in buoyancy, while the process

remains similar. Using a pre-calculated flow field to guide particle movement simplifies the simulation by treating particles as independent entities, eliminating the need for inter-particle calculations. This not only reduces significantly the overall execution time but also offers users high flexibility over the quality of the simulation and simplify the implementation.

6.3.2 Erosion process



Every time the particle hits the ground, a given amount $q_{\text{detachment}}$ of sediment is detached from the ground (red arrows) while another amount q_{deposit} of sediments is deposed at this location (green arrows). Our erosion model is based on the work of Wojtan et al where regular 3D grids are used to estimate the fluid velocity and sediment transport (Wojtan et al., 2007). In the spirit of (Krištof et al., 2009), we transposed their method into a particle-based erosion simulation, but, in our proposition, we decouple the particle system from the fluid simulation, making the process more flexible and opening the door for richer effects that can easily be produced.

Detachment

As a particle approaches the surface of the terrain, its motion applies friction at the interface between fluid and ground, causing bedrock to dislocate microscopic parts, that we call abrasion. We use pseudoplastics model to approximate the amount of matter removed due to the shear forces while considering the physical properties of the fluid and the ground (Wojtan et al., 2007).

The shear rate θ is approximated by the relative velocity of the fluid to the solid boundary v_{rel} over a short distance l. We approximate the shear stress τ at the solid boundary by a power-law:

$$\tau = K\theta^n \tag{6.1}$$

where $\theta = v_{rel}/l$, K is the shear stress constant (often set to 1) and $n \in [0, 1]$ is the flow behaviour index. Shear-thinning models typically assume n close to $\frac{1}{2}$, which is why we used this value as a constant.

We can then compute the erosion rate ε at any contact point between a fluid and a solid boundary using Equation (6.1) by

$$\varepsilon = K_\varepsilon(\tau - \tau_{\text{critical}})^a \tag{6.2}$$

with $K_\varepsilon \in [0,1]$ a user-defined erosion constant, τ_{critical} the critical shear stress value for which the matter starts to behave like a fluid and a a power-law constant, typically considered as $a = 1$.

In our method, the eroded quantity is approximated as the material contained in the half sphere, of radius R, in the normal opposite direction at the particle impact point (Figure 6.6). We then use Equation (6.2):

$$q_{\text{detachment}} = \varepsilon \frac{2\pi R^3}{3} \tag{6.3}$$

to get the final eroded amount $q_{\text{detachment}}$. The particle is also defined by a maximal amount of sediments that can be contained in its volume before being saturated noted $C_{\max}$. Note that this constant will be used for the settling velocity computation Equation (6.7).

Deposition

The eroded sediments are considered in suspension in a fluid and are affected by its velocity. A fluid particle then transports the sediments in its flow until gravity settles it onto the ground again. The effect of gravity is modeled by a settling velocity w_s defined in Eq Equation (6.7). We consider that the amount of sediment settled is proportional to the norm of the settling velocity as proposed in (Wojtan et al., 2007) with $\omega \in [0,1]$:

$$q_{\text{deposit}} = \omega ||w_s||. \tag{6.4}$$

6.3.3 Transport

Our simulation is computed by integrating the full trajectory of multiple particles at each iteration unlike most other erosion methods. This allows to constantly have a terrain in a plausible state, while giving the possibility to increase the aging effect by running more iterations. Note that, reducing progressively the overall erosion strength can be used as a strategy to adapt the computation time to a chosen level-of-details.

We first present how to compute the particle speed using particle's physics then how to add optional medium velocity field to add a fluid simulation or user control.

Particle's physics

From its independence with other particles: we consider each particle following Newton's laws of motion.

First, we define the external forces $\vec{F}_{\text{ext}}$ applied on each particle, we consider gravity and buoyancy. We calculate the buoyancy force $\vec{F}_{\text{buoyancy}} = -\rho_{\text{fluid}} V \vec{g}$ with ρ_{fluid} the density of the fluid, V the volume of the particle and $\vec{g}$ the gravitational acceleration, but we can also calculate the force of gravity $\vec{F}_{\text{gravity}} = m_{\text{particle}} \vec{g}$ with m_{particle} the mass of the particle. We then have the final external force $\vec{F}_{\text{ext}} = \vec{F}_{\text{gravity}} + \vec{F}_{\text{buoyancy}} = m_{\text{particle}} \vec{g} - v_{\text{fluid}} V \vec{g}$ knowing the density of an object $\rho_{\text{particle}} = \frac{m_{\text{particle}}}{V}$, we have:

$$\vec{F}_{\text{ext}} = V \vec{g} (\rho_{\text{particle}} - \rho_{\text{fluid}}). \tag{6.5}$$

The particle velocity v can be integrated from Equation (6.5) by:

$$v = \int \vec{F}_{\text{ext}} \, dt + w_s + v_0, \tag{6.6}$$

with w_s the settling speed of sediments in a fluid with a viscosity μ given by Stoke's Law (Stokes, 1850):

$$w_s = \frac{2}{9} \vec{g} R^2 \frac{(\rho_{\text{particle}} - \rho_{\text{fluid}})}{\mu} f(C). \tag{6.7}$$

We use the Richardson-Zaki relation as the hindered settling coefficient:
$f(C) = 1 - \left(\frac{C}{C_{\text{max}}}\right)^n$
with C and C_{max} respectively the fraction of volume of sediments contained and the maximal fraction of sediments the particle can contain, and n an exponent typically 4–5.5, which we set to 5 (Richardson and Zaki, 1954; Wojtan et al., 2007).

Finally, the particle position can be integrated as:

$$\mathbf{p} = \int v \, dt + \mathbf{p}_0.$$

When the particle hits the ground, a coefficient of restitution affects its behaviour by reducing its velocity post-collision. This value depends on ground material as it is influenced mainly by the material's particle shape, coefficient of friction and density (Yan et al., 2020). Less bouncy particles lose speed quickly and settle down sooner, forming a steeper pile (Figure 6.4 blue), or a higher talus angle like chalk. On the other hand, more bouncy particles disperse more widely upon hitting a surface, resulting in a gentler accumulation like clay (Figure 6.4 red).

Velocity field

In our model, we allow the user to add a velocity field to the environment that influences particles motion. This velocity field can be the result of a complex

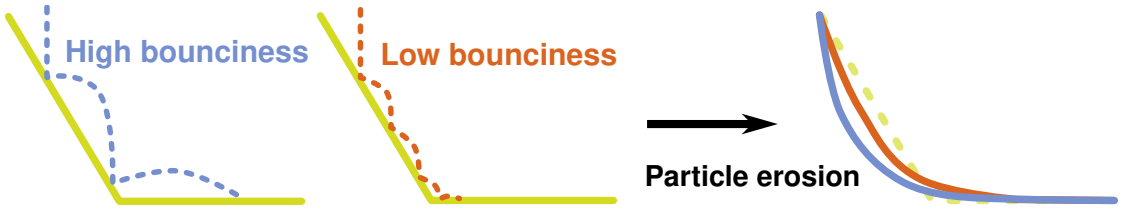


Figure 6.4: The coefficient of restitution affects the amount of energy absorbed from the particle when hitting the ground. Here, rain is applied on an initial slope (yellow). Only two particles are displayed, with a high (blue) and low (red) coefficient of restitution. The resulting slope after erosion is displayed in blue and red (right).

fluid simulation, a uniform vector field, or an artistic motion field. We modify Equation Equation (6.6) such that the particle's speed will be influenced by the velocity field as follows:

$$v = \int \vec{F}_{\text{ext}}\, dt + w_s + \alpha v_{\text{fluid}} + v_0, \tag{6.8}$$

with v_{fluid} medium velocity field modulated by $\alpha \in [0, 1]$.

Our particle system can model intricate scenarios, like the erosion caused by water currents on the seabed or aeolian erosion. The velocity field remains static during the erosion, which may cause inconsistencies in the fluid velocity field. However, minor changes can be overlooked to maintain a balance between realism and computational efficiency (Tychonievich and Jones, 2010). We offer several velocity improvement methods:

- **Fluid simulation refinement:**
  Many erosion systems incorporate fluid simulation, requiring regular updates for erosion and velocity (Krištof et al., 2009; Wojtan et al., 2007). Our method can use fluid simulations with multi-resolution refinement, with the possibility to focus the velocity field adjustments near the updated boundaries of the surface (Roose et al., 2011).

- **Particle velocities in fluid simulation:**
  With a Lagrangian fluid simulation relying on particle systems (Koschier et al., 2022), our particle velocities can be incorporated in its computation. This approach is only a provisional solution due to potential parameter mismatches with main fluid simulation.

- **Velocity field diffusion:**
  Given the minor changes to the surface level at each erosion iteration, which reflect the gradual alterations in terrain surface, we can estimate that the velocity at a fixed point transitioning between the inside and outside of the terrain closely mirrors the velocities observed in its surrounding area. In this context, we can simply interpolate the velocity field at any transitioning point. This simple method, as used in Figure 6.8, allows us to find a balance between achieving realistic flow simulations and maintaining computational efficiency.

6.4 Our erosion method

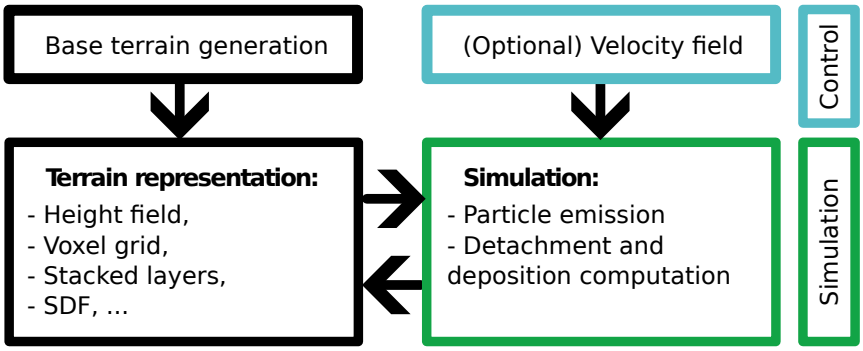


Figure 6.5: Our overall pipeline: our erosion process compute matter displacement of a terrain using an arbitrary representation as long as intersections between particles and the ground can be detected. An optional velocity field, provided by the user, guides the particles trajectories. We propose surface alteration methods to apply the erosion to the terrain in a coherent way between possible representations.

In this section, we describe how to apply detachment and deposition to different terrain representations with our method (Figure 6.5). We cover the most commonly used representations namely height fields, layered terrains, voxel grid and implicit surfaces, note that our work could be extended to additional representations. Two conditions need to be satisfied for a representation to be eligible for our erosion method: being able to evaluate the intersection of a particle with the ground and compute the normal of the terrain at this point. To the best of our knowledge, all representation do.
We use Verlet integration for the particle's physics (Verlet, 1967), with low error rate and stability even for high dt, reducing computation time for negligible imprecision (Baraff and Witkin, 1998; Swope et al., 1982).

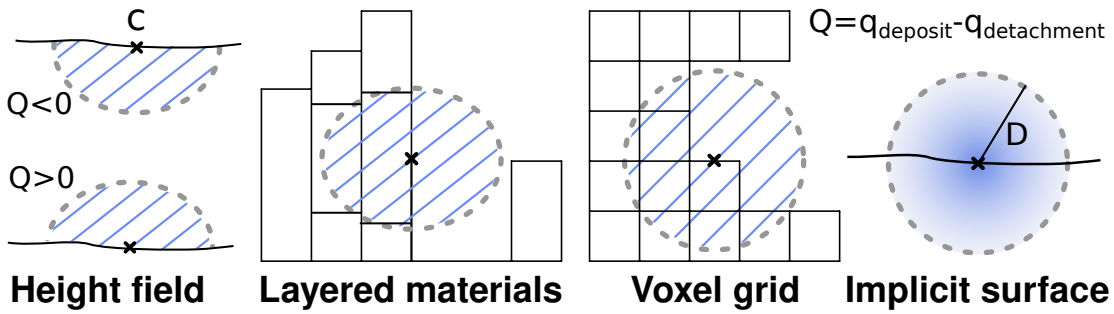


Figure 6.6: Illustration of the material detachment in the (half-)sphere at contact point C (cross) on different representations. (height field) When $Q < 0$ material detachment happen in the bottom scaled half sphere of the particle's contact with the ground, while the deposition is applied on the upper half sphere of volume when $Q > 0$. Unlike the height field, for 3D terrains detachment and deposit are applied in the full sphere around the contact point.

For all the representations, the amount of material absorbed by the particle, i.e. the erosion value $q_{\text{detachment}}$ from Equation (6.3), is taken around the particle at a radius R, meaning that the modification of the terrain by a particle at position c will only occur for the positions $\mathbf{p}$ satisfying $||\mathbf{p} - c|| < R$. At the same time, the amount q_{deposit} from Equation (6.4) is deposited, resulting in a change $Q = q_{\text{deposit}} - q_{\text{detachment}}$
In our simulation, while the dynamics are informed by physical principles, the particle size is conceptualized within a dimensionless framework. This provides the flexibility to adapt our results to various real-world scales, ensuring the applicability of our model across diverse scenarios. Note that, for a 2.5D terrain, we can consider that half of the sphere surrounding the particle is affected which has a volume of $V_{2.5D} = \frac{2\pi R^3}{3}$ while a 3D terrain is affected by the full sphere $V_{3D} = \frac{4\pi R^3}{3}$ (as illustrated Figure 6.6). In the following sections, we will describe the strategies used to modify the amount of matter for different representations.

6.4.1 Application on height fields

On a height field defined by $h(\mathbf{p}) = z$, the intersection point with the surface is verified at $\mathbf{p}_z = h(\mathbf{p})$, and the normal can be computed at the intersection point.

For this representation, the half sphere is scaled in the z direction to fit $\alpha V = Q$ using $\alpha = \frac{Q}{V}$. We then can decrease the height $h'(\mathbf{p})$ at all points $\mathbf{p}$ by the height of the scaled half sphere at position $\mathbf{p}$. Given the height of the scaled

half sphere of center c and the distance of the particle to the center $d = ||\mathbf{p} - c||$ by $h_{\text{half sphere}}(\mathbf{p}) = \alpha\sqrt{R^2 - d^2}$ for all $\mathbf{p}$ such that $d \leq R$ the radius around a particle.

This change of height can be sampled at all points of the 2D grid by reducing the height by

$$\Delta h(\mathbf{p}) = \frac{\sqrt{R^2 - d^2}}{\alpha} = \frac{Q}{\frac{2}{3}\pi R^3}\sqrt{R^2 - d^2} \tag{6.9}$$

The height at each point after an erosion is then computed as $\tilde{h}(\mathbf{p}) = h(\mathbf{p}) + \Delta h(\mathbf{p})$.

6.4.2 Application on layered terrains

Layered terrains are defined as $\mu : \mathbb{R}^3 \mapsto \mathbb{N}$ assigning a discrete material index μ for any point in space (Beneš and Forsbach, 2001; Peytavie et al., 2009). In the original work, outer borders stack elements of the terrain are transformed into density-voxels to enable global erosion through height changes. We enable the erosion/deposition process directly on the layers hence removing the need for representation changes.
When intersecting the terrain, the amount eroded for each material stack should be the integration of the volume of the intersection between the sphere surrounding the particle and the cubicle represented by the stack. Since there is no easy solution (Jones and Williams, 2017), we approximate the volume of the stack we need to alter using the previously defined height field equation Equation (6.9). At a distance d from the particle, the height is defined as:

$$H(d) = \frac{|Q|}{\frac{2}{3}\pi R^3}\sqrt{R^2 - d^2}. \tag{6.10}$$

If $Q > 0$ (more deposition is applied that detachment) then we transform the materials in the stack contained in the sphere to become ground material. For $Q < 0$ the materials are transformed in background material.

6.4.3 Application on implicit terrains

Implicit terrain are defined using a function $f(\mathbf{p})$ and its variation resulting from the erosion process using $\Delta f(\mathbf{p})$. We propose a strategy to compute $\Delta f(\mathbf{p})$ at any point of the sphere surrounding the erosion point based on metaball

primitives. At each contact point a metaball is added to create a hole or a bump in the terrain. A metaball is defined as:

$$\Delta f(\mathbf{p}) = \frac{3Q}{\pi}\frac{(1-d)}{R} \tag{6.11}$$

with d the distance of the point $\mathbf{p}$ to the sphere center. For all point $\mathbf{p}$ for which $d \geq R$, $\Delta f(\mathbf{p}) = 0$ (see **??**).

As they are the most commonly used representations, we propose a formulation to erode implicit terrains defined by Signed Distance Functions (SDF) and by gradient or vector fields.

Signed Distance Functions

Considering SDF, the terrain is defined as the 0-set of the signed distance function $f : \mathbb{R}^3 \mapsto \mathbb{R}$, hence, for $f(\mathbf{p}) = 0$, the inside as $f(\mathbf{p}) < 0$ and outer-part (i.e. air or water) as $f(\mathbf{p}) > 0$.
The particle erosion applies at impact points at discrete positions, so we propose to add or subtract metaballs defined using equation Equation (6.11) to respectively deposit or erode material using a composition tree:
$metaball(\mathbf{p}) = -\Delta f(\mathbf{p})$.
Now the eroded terrain function $\tilde{f}(\mathbf{p})$ will be evaluated at each point $\mathbf{p}$ from the initial terrain value $f(\mathbf{p})$, the erosion function $metaball(\mathbf{p})$ and the composition function $g(f_1, f_2)$:

$$\tilde{f}(\mathbf{p}) = g(f(\mathbf{p}), metaball(\mathbf{p})).$$

As a metaball is added for each particle bounce on the terrain space partitioning optimization algorithms such as k-d trees, BSP trees or BVH can easily be used to improve performances.

Other implicit terrains

are present in the literature, notably a 2.5D representation based on the surface gradient (E. Guérin et al., 2022) and a 3D representation based on curves (Becher et al., 2017) for which the trajectory of each particle projected to the closest surface could be used to define the alteration of the terrain.
In the case of gradient-based representation, we propose to use the partial derivative from the equation of the 2D scalar fields Equation (6.9) that gives:

$$\nabla h' = -\frac{Q}{\frac{2}{3}R^3}\frac{1}{\sqrt{R^2 - d^2}}\vec{CP} \tag{6.12}$$

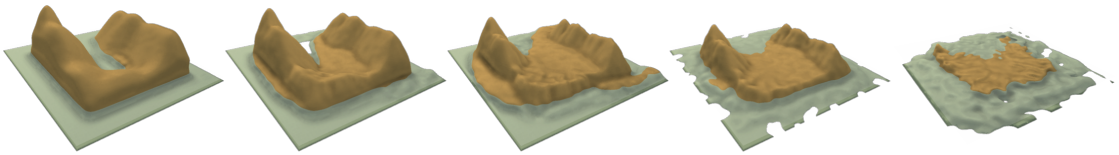


Figure 6.7: Our erosion method is applied iteratively on a completely synthetic island, the terrain is altered to obtain a plausible shape by forming rills. The use of particles with hydraulic densities dropped from the sky results in a strong erosion on the sides of the mountains, and the particles that slide to the sea are mainly drifting offshore resulting in the formation of small beaches and a weaker erosion on the bottom of the water body. Repeating the process causes the island height to decrease progressively up to the point where only the submerged part of the terrain is sheltered from erosion.

with $\vec{CP}$ the vector from the position $\mathbf{p}$ to evaluate to the center of the erosion point c. Now the new gradient field can be computed as:

$$\nabla\tilde{h}(\mathbf{p}) = \nabla h(\mathbf{p}) + \nabla h'(\mathbf{p}).$$

6.4.4 Application on voxel grids

We consider two of the voxel grids representations: density-voxel grids and binary voxel grids for which we present our material alternation strategy.

Density voxels

We consider "density-voxel" grids defined on $f : \mathbb{Z}^3 \mapsto [-1, 1]$ for which a voxel is be full for $f(\mathbf{p}) = 1$, partially full for $-1 < f(\mathbf{p}) < 1$ or empty for $f(\mathbf{p}) \leq -1$. This definition allows us to erode them smoothly. Since this kind of grid is a discretizaion of a scalar function, We could directly use Equation (6.11), as described previously, but we take advantage of the discrete nature of the representation to avoid expensive computation.
We apply the erosion from a particle at position c on all points $\mathbf{p}$ in the volume proportionally to the distance from the center of the sphere $d = ||\mathbf{p} - c||$ to find an approximation to the real erosion value per voxel $Q_{approx} = Q\frac{1-d}{R}$. Using their discrete nature, we rectify this value to sum up the total erosion value to Q by dividing each value by the sum of the distances. We now consider eroding the "empty" voxels since their density can drop until -1. We then have for all

surrounding voxels:

$$\Delta f(\mathbf{p}) = Q\frac{(1-\frac{d}{R})}{\sum(1-\frac{d}{R})}. \tag{6.13}$$

Resulting voxel value is computed as $\tilde{f}(\mathbf{p}) = f(\mathbf{p}) + \Delta f(\mathbf{p})$. In our implementation, when $f(\mathbf{p}) > 1$, we simply transport the density excess to the above voxel, giving it a very close analogy to height fields as long as $|\Delta f| < 1$.

Binary voxels

The terrain can be represented using an occupancy function as $f : \mathbb{Z}^3 \mapsto \{0, 1\}$ where a voxel $f = 1$ defines the ground and $f = 0$ the background.
We propose to apply particle erosion by assigning voxels a number of hits, and transform them as air or as ground when this number reaches a critical value C that is proportional to the particle's strength parameter K_ε (Beardall et al., 2010). On a hit, all voxels in a radius R receive a hit number:

$$\Delta hits = \lfloor \alpha \Delta f \rfloor \tag{6.14}$$

with Δf the erosion per voxel computed using Equation (6.13) and α a coefficient high enough to obtain values above 1.
All voxels with $\#hits > C$ are transformed to background and voxels with $\#hits < -C$ are transformed to ground.
Note that, a binary voxel grid can also be transformed into a density-voxel grid to be eroded smoothly.

Our formulation for height fields Equation (6.9), can be used to erode 2D scalar field-based representations. Similarly, our proposition for SDF Equation (6.11) enables erosion for continuous 3D scalar fields and voxels Equation (6.13) for discrete 3D scalar fields respectively.

6.5 Results

Our erosion process enables the simulation of a wide range of erosion effects on the major terrain representations alike. In this section, we present applications that demonstrate the versatility of our method by changing the particle's effect size, quantity, density, maximum capacity, deposition factor and the velocity fields. The results of each process are presented in Figure 6.14, parameters used are available at Table 6.1. It is important to note that all erosion examples presented in this section are available for any 3D terrain representation.

Name	Rep.	Dimensions	Res	$\#P$	$\#N$	R	COR	ρ_{particle}	C_{factor}	ε	ω
Rain	H	100x100	20	100	10	1.0	1.0	1000	10.0	2.5	0.3
Coastal	DV	100x100x30	10	80	3	5	0.1	500	10.0	5.0	0.5
Meanders	I	N/A	N/A	10	20	5.0	1.0	1000	1.0	1.0	1.0
River	H	100x100	5	100	50	1.5-5	0.5	900	0.1	1.0	1.0
Landslide	H	100x100	20	200	10	2.5	0.2	500	0.1	1.0	1.0
Volcano	DV	100x100x40	50	150	30	1.0	5.0	2000	1.0	1.0	5.0
Karst	BV	100x100x50	2	1000	40	5	0.5	500	10.0	5.0	0.5
Tunnel	DV	100x100x50	1	100	100	2.5	0.1	500	1.0	1.0	1.0
Wind	DV	100x100x50	0.2	100	10	1.5	0.9	1.5	1.0	1.0	1.0
Underwater	H	100x100	10	100	50	2.5	0.9	1000	1.0	1.0	1.0

Table 6.1: Parameters used for the generation of the terrains presented in Figure 6.14, with "Rep" the representation (H: Heightmap, DV: Density-voxels, BV: Binary voxels, I: Implicit) "Res" the resolution in meter per voxel or cell, $\#P$ the number of particles per iteration, $\#N$ the number of iterations, R the particles radius (in voxel or cell unit), COR the coefficient of restitution, ρ_{particle} the particle density in $\text{kg}\,\text{m}^{-3}$, C_{factor}, ε and ω respectively the capacity, erosion and deposition factors, "Vel field" the type of velocity field used and t the computation time of the simulation in seconds on CPU.
(1) The velocity field is a vector field defined as $v_{\text{fluid}}(\mathbf{p}) = [0\ sin(\mathbf{p}_x)\ 0]^T$.

However, we cannot create volumetric structure, such as overhangs, using 2.5D representations (height fields).

Environment density ρ_{fluid} is set to $1\,\text{kg}\,\text{m}^{-3}$ above water level (terrain blue part) and to $1000\,\text{kg}\,\text{m}^{-3}$ below it. Velocity field's refinement is done by using the presented diffusion strategy.

6.5.1 Rain

Hydraulic erosion from rain is the most common process used in terrain generation. In this case, particles are seen as water droplets falling from the sky and rolling downhill due to the gravitational force of Earth. No velocity field is required from fluid simulation. These parameters result in a detailed geometry of the rills on the side of mountains that quickly emerge and deposit many sediments in the valley. We demonstrate the result of rain erosion in *Figure 6.14: Rain* with a computation time of 4 seconds.
Using this erosion parameters in combination with water bodies results in different outcomes (Figure 6.7). The terrain above water is directly affected by the erosion process while particles colliding with the underwater part of the terrain

are slowed down and filled with sediments, leading to mainly apply deposition. The result is a typical hydraulic erosion on mountains and the formation of slopes and beaches near water level.

6.5.2 Coastal erosion

Waves repeated motion creates coastal erosion, that can be seen as cliffs with holes at the water level.
We apply a uniform velocity field in the water pointing towards the coast to simulate waves and emit particles from the water area with a large size, a density between air and water densities, a high capacity factor and a low deposition factor ω. Using these parameters, the erosion process is focused at the interface of air and water, and apply a coarse detachment while depositing a very small quantity of sediments, simulating the corrosive effect of water on limestone.
This effect can only be simulated on 3D terrain representations, but will create cliffs on a 2D representation. *Figure 6.14: Coastal* presents the result of coastal erosion on a density-voxel grid that creates overhangs around sea level using a small amount of particles. Note that, the same effect using an alternate implicit representation based on SDF is displayed in Figure 6.10. A shaded version of this effect is presented in Figure 6.1.

6.5.3 Rivers

Given a source point, we generate particles that run downhill, simulating the formation of a river. More complex erosion simulation using fluid simulations like SPH (Krištof et al., 2009) would create realistic results at the cost of high processing time. Our method offers the flexibility to be applied either with a velocity field (simple, used given or resulting from a fluid simulation) or without allowing for simplicity and efficiency.
When provided with a hand-made or procedural velocity field, our particle system can reproduce simple river meanders (*Figure 6.14: Meanders*).
Figure 6.14: River presents a river that has been modeled by emitting water particles with different sizes that ranges from 1.5 m to 5 m, a high coefficient of restitution and a low capacity factor. Random sizes are used to simulate a river for which the flow rate had fluctuated over formation time, while the low capacity ensure that the banks of the river stays smooth. A high coefficient of restitution is a strategy that let the particles flow with low friction, approaching a water behaviour. Our particles are affected only by gravity, without fluid simulation.

6.5.4 Landslide

are mainly caused by large amount of water saturating the ground and flowing downhill, transporting matter in its path.
By using water particles with a medium size, a low coefficient of restitution and a low capacity factor but a high deposition factor ω, they transport sediments on short distances as the velocity quickly drops to 0, and ground material is completely spread along its path since it is easier to deposit the same amount of sediment than the eroded amount at each collision point. Reducing the density of the particle simulates a rise of viscosity in the settling velocity formula, increasing again the quantity of matter to deposit at contact with the ground. By this means, we can simulate landslides as illustrated on *Figure 6.14: Landslide*. A smoother surface is resulting, compared to the rain erosion as the rills are filled with sediments as soon as they begin to form. By setting the initial capacity of the particle equal to 10% of its max capacity, the mass of the terrain increases, simulating a volcano eruption as illustrated on *Figure 6.14: Volcano*.

6.5.5 Karsts

networks are created over hundreds of years from the corrosion of water on the limestone in the ground. A limited number of methods have been proposed for the procedural generation of karsts (Paris et al., 2021).
By reducing the deposition factor ω, the particles simulate corrosion (without mass conservation). We can use the same particle parameters than the coastal erosion (big size, a density between air and water densities, a high capacity factor and a low ω) and optionally provide a 3D shear stress map. The karst will automatically follow the softest materials, which is geologically coherent as given in example in *Figure 6.14: Karst*, where we can observe a "pillar" that is formed in the center, and thus the karst forms two corridors that finally merge partially. Underground results are only available for representations allowing 3D structures. Another underground terrain simulation is shown in *Figure 6.14: Tunnel* in which a water runoff is eroding a tunnel without the use of a fluid simulation. Here, when particles bounce often on the terrain surface, the coefficient of restitution may be seen as a viscosity parameter.

6.5.6 Wind

erosion is a significant process in desertscapes shaping since there are no obstacles on the airflow path. Air particles can reach high velocities, transporting sand over long distances forming either dunes or are blasted into rocks, eroding into goblins.

By setting the density of our particles close to $1\,\mathrm{kg\,m^{-3}}$, two erosion simulations can be applied at once. Air particles follow closely the flowfield given by the user in air. This flowfield can be given from a complex simulation, a user-defined wind rose (Paris et al., 2019a) or a random flowfield with a general direction. The generation of the different sand structures depends on the velocity field provided, and a simple field will easily generate linear dunes. On contact with a rock block, the simulation will automatically erode block borders, creating shapes looking like gobelins.
Figure 6.14: Wind gives an example of wind erosion on a flat surface with rock columns being eroded. Given a strong 2D velocity field computed by the high wind simulation proposed in (Paris et al., 2019a) is used on light particles, the simulation is fast thanks to the low number of collisions each particle has with the ground.

6.5.7 Underwater currents

Procedural generation of underwater 3D terrains has received little attention. The difference between the underwater and the surface rely on the buoyancy force that is much stronger, meaning that the water flow has a much more impacting effect on erosion than wind. Taking into account the density of the environment and the velocity field of water in our formulas are the keys to be able to apply any erosion in this environment. Our method works in a water environment by giving at least water density to particles. Given a velocity field describing underwater currents from a complex simulation or from a sketch, the particle system erodes the terrain.
In the example presented in *Figure 6.14: Underwater*, the velocity field is given by a simple 3D fluid simulation (Stam, 1999) applied on the terrain.

A complex water flow simulation is computed using SIMPLE (Caretto et al., 1973) fluid simulation with OpenFOAM. The resulting erosion can then follow complex water movement and erode the terrain at the most affected parts of the 3D terrain as the trajectories of the particles (green) is highly affected by the fluid velocity (blue). The density of the particles and the environment being close, the buoyancy cancels most of the gravity force, leaving the velocity of the particles computed by the fluid velocity v_{fluid} and settling velocity w_s from Equation (6.8) (Figure 6.8).

6.5.8 Multiple phenomena

A terrain eroded with multiple erosion phenomena applied on a 500x500x50 density-voxel grid is illustrated in Figure 6.9. Here, water-density particles are

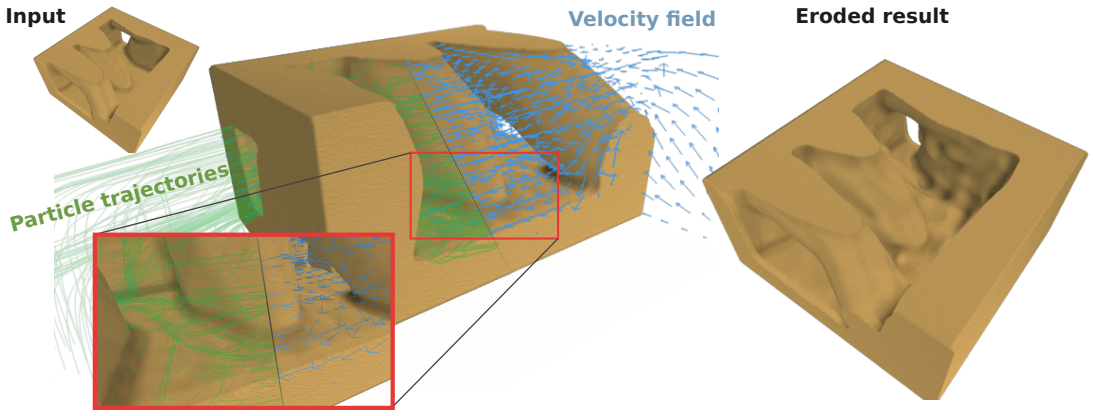


Figure 6.8: A complex water flow simulation is computed using OpenFOAM. Particle trajectories (green) are highly affected by the fluid velocity (blue). Most the terrain exposed surfaces is eroded (bottom).

applying rain on the terrain while the coasts of the river are being eroded thanks to a velocity field defined at the water level. The velocity field defined in the air mainly affects particles with air-density, such that wind erosion can be applied at the same time. The computation of these effects took 7 seconds on CPU.

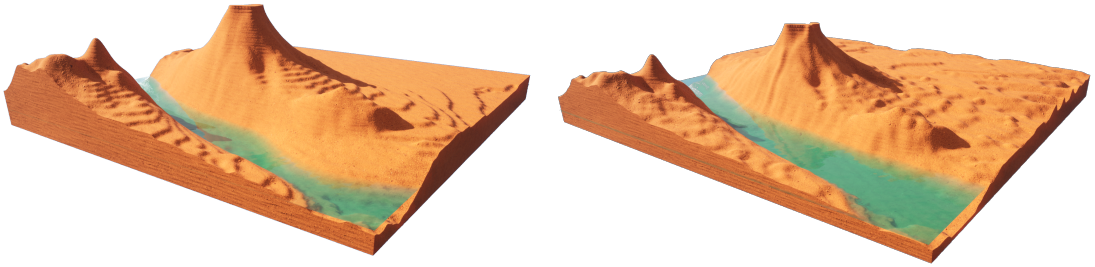


Figure 6.9: Multiple erosion types can be combined. On an initial synthetic 500x500x50 density voxel grid, the a wind erosion is applied on the surface of the terrain while hydraulic erosion shapes the rills and the base of the mountains. A water current digs its borders and spreads sediments at the bottom.

6.6 Comparisons

In the following section, we compare our method with existing ones to show that while we are versatile on the terrain representation, we are also able to

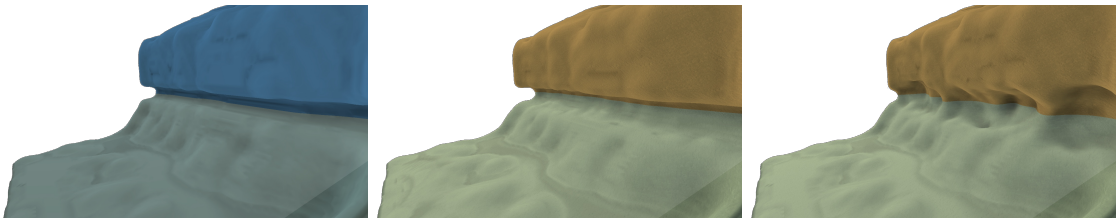


Figure 6.10: The algorithm proposed by (Paris et al., 2019b) allows for the simulation of coastal erosion (left) that we can reproduce almost identically by allowing our particles to collide only once with the ground and applying only erosion (center). If we apply our erosion with the full tracking of our particles and using deposition, we can achieve more diverse results (right).

reproduce various effects without applying specific algorithms. The other works are displayed in blue to distinguish them from ours.

6.6.1 Coastal erosion on implicit terrain representation

Paris et al., *Terrain Amplification with Implicit 3D Features* (2019) present an erosion simulation method applied to implicit terrains able to create coastal erosion, karsts and caves by adding negative sphere primitive in the terrain's construction tree. The positions of the spheres are determined using a Poisson disk sampling at the weakest terrain area defined by the Geology tree of their model. They are simulating the corrosion effect of water on the rocks. Our work is also able to approximate this phenomena by defining the position of these sphere primitives at the position where the water particles hit the surface. While the computation time of the positions of the sphere is higher due to the fact that we are evaluating the position of our particles at every time step in the implicit model (which could be improved by the triangulation of the implicit surface, or better, a dynamic triangulation), the distribution of our erosion primitives is based on a physical model instead of a mathematical model, meaning that we can integrate more easily the direction and strength of the waves for example. The management of their sphere primitives can be replicated with our method by considering that a particle exists until a collision occurs, at which point it disappears. Their method is not conserving the mass of the terrain, which is acceptable for the corrosion simulation, but limits its validity for other erosion simulations. In our method, the particle can be tracked until it settles, ensuring mass conservation (Figure 6.10);

6.6.2 Wind erosion on voxel grid representation

Beardall et al., *Directable weathering of concave rock using curvature estimation* (2010) propose a weathering erosion on voxel grids by approximating and eroding continuously the most exposed voxels. When a solid voxel is decimated, it is considered deposit and is displaced down the slope until a minimal talus angle in the terrain is reached and if the deposition is eroded again, it disappears. Our work is able to reproduce their algorithm by sending our particles from a close distance to the terrain surface. By doing so, we reproduce the erosion process as much as the deposition process since the air particles, filled with sediments, is falling automatically towards the local minimum of the erosion point. Just like in their work, we can easily define the resistance value of the materials to add diversity in the results. By adding the possibility of a wind field, even a very simple uniform vector field, to the simulation, we naturally add the wind shadowing effect that protects a gobelin surrounded by bigger gobelins, and also allows the deposit slope to fit more closely to the wind direction (Figure 6.11).

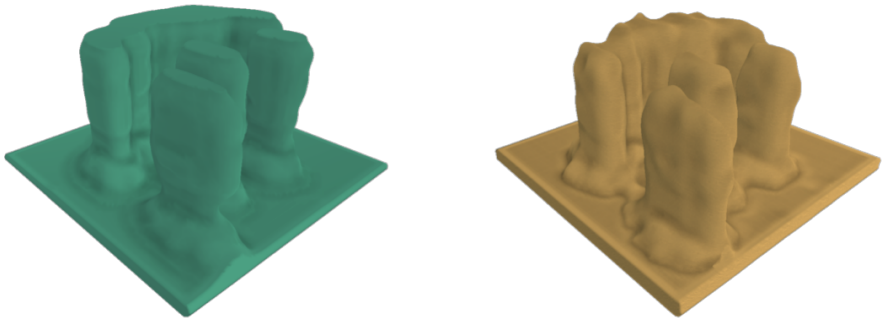


Figure 6.11: The algorithm proposed by (Beardall et al., 2010) allows for an efficient simulation of the spheroidal erosion, making the creation of gobelins on voxel grids in a plausible way (left). Our algorithm naturally erodes the most exposed areas of the terrain when particles are affected by the wind (right).

6.6.3 Hydraulic erosion on height field representation

Mei et al., *Fast hydraulic erosion simulation and visualization on GPU* (2007) integrate and adapt to the GPU the pipe model proposed in (O'Brien and Hodgins, 1995) for the fluid simulation. This simulation is simple but efficient enough to approximate the Shallow-Water equations in real time and use the speed of columns of water to compute the erosion and deposition rate on the 2D grid of the terrain at each time step. Using columns of water even allows the flow

to overpass small bumps on the terrain over time. Our method initially rely on a stable fluid flow that is consistent during the whole life time of a particle, but by refining the simulation at each time step instead of at the end of the particles lifetime, our erosion model is able to reproduce this effect, allowing the terrain to have a single batch of fluid going through it. Our method can be seen as a generalization of Mei at al. that can then be used on more than discrete 2D grids (Figure 6.12).

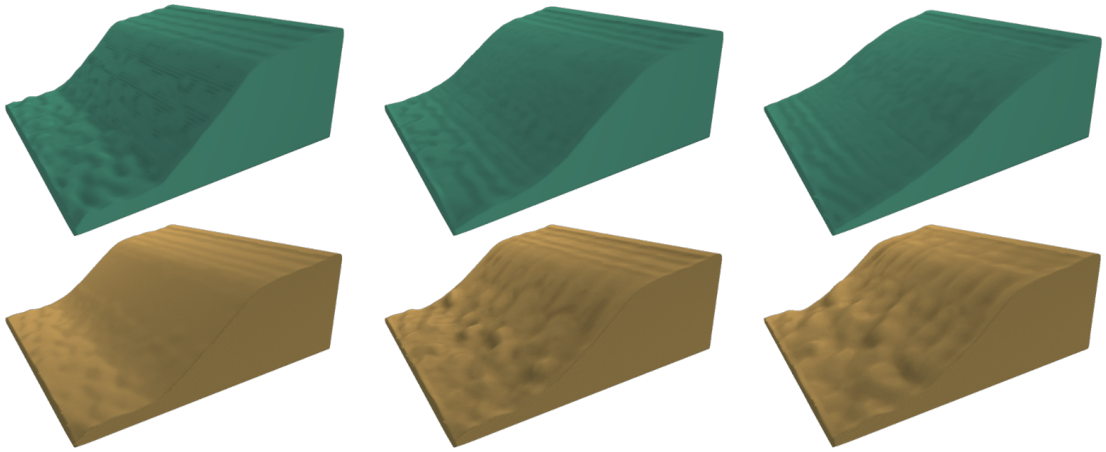


Figure 6.12: While our resulting geometry on the hydraulic erosion (bottom) is less smoothed than the one proposed proposed by (Mei et al., 2007) (top), our method allows the application on more terrain representations than the height fields only.

6.6.4 Wind erosion on stacked materials representation

Paris et al., *Desertscape Simulation* (2019) simulate the effect of wind over sand fields defined on stacked materials, creating dune structures, even taking into account obstacles like (Roa and Benes, 2004) and different material layers like vegetation (Cordonnier et al., 2017a) that are not affected by abrasion(Paris et al., 2019a). A wind field simulation is required to produce results, and while (Roa and Benes, 2004) and (Onoue and Nishita, 2000) consider a uniform vector field, this work consider a dynamic vector multi-scaled warped field from the terrain height. The sand grains then apply multiple moves: sand lift, bounces, reptation and avalanching. Once the sand is lifted by the wind, the trajectory of the grains can be seen as the displacement of particles, fitting completely with our model as illustrated Figure 6.13.

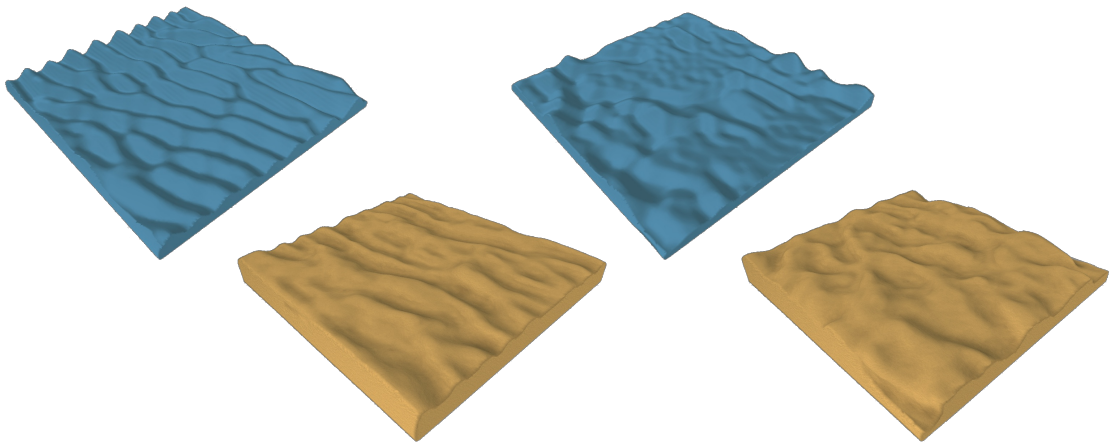


Figure 6.13: The algorithm from Paris 2020 allow the generation of desertscapes (top), which we can (at least partially) reproduce with our erosion simulation (bottom). The different effects are achieved by affecting the wind direction and strength.

6.7 Discussion

This work is a generalization of erosion that is applicable to any terrain representation. In practice, while similar particle physics is used on different terrain representations, using similar parameters does not ensure resulting in the same eroded terrain. Surfaces and normals being approximated differently have rippling effect on particle trajectories. Note that, not all effects can be applied to all representations, forinstance, karsts generation on 2.5D data structures.

6.7.1 Realism

Realism of the erosion simulation is highly correlated to the size and quantity of particles used and their distribution. Using too few or distributing them too sparsely will result in a terrain that is unrealistic since the alteration will have localized effects, breaking process homogeneity.
The resolution is also limited by the number and size of the particles, which can be problematic on implicit terrains that can theoretically have a infinite resolution.
Our method allows to perform erosion on implicit terrains. However, in its current form, our algorithm is time expensive on implicit representations since a large number of primitives are added in the composition tree. Using skeletons-defined primitives (Hong, 2013; Rigaudière et al., 2000)from particles trajectories

and erosion/deposition values could be a solution to optimize the computation time.

6.7.2 Usage of velocity fields

In our erosion algorithm, we simplify particle physics to enhance computational efficiency and facilitate parameterization. We use the velocity field from fluid simulations to approximate particle velocities. Sediment mass is harnessed to compensate for this approximation, allowing compatibility with various fluid simulation algorithms. Velocity fields can be recomputed at a frequency meeting the applications needs, ranging from "classic erosion simulation" (recomputed at each time step) to "simple simulation" (never recomputed). We addressed provisional adjustments to mitigate discrepancies when terrain changes due to erosion are not reflected in a static velocity field in section “3.3 Transport”. However, it is important to note that these are expedient solutions and may not fully capture precise dynamics of an evolving terrain.

6.7.3 Performances

To facilitate parallelization, we intentionally overlook particle interactions and sediment exchanges, albeit at the expense of achieving smoother results. Surface collisions are simplified to basic bounces with a damping parameter instead of relying on complex particles and ground properties (Young’s modulus, friction, material, ...)(Yan et al., 2020), further easing the parameterization process. However, these simplifications, combined with the inherent discrete nature of particles, as opposed to the continuous nature of erosion, result in a correlation between realism and particle count.
The performance of our method is influenced by the time required for collision detection. Consequently, we mainly observe better performances with explicit terrain models than with implicit models.

Particle’s atomicity

While we can replicate various effects, the "fan" shape commonly observed in natural erosion patterns is not perfectly represented. This limitation arises because we do not account for the splitting of a particle, a process that significantly influences the multidirectional dislocation and trajectory of individual particles (Ranz et al., 1960). Additionally, we acknowledge an issue where particles may collide with the ceiling and the deposition is stuck. While a potential resolution involves splitting particles upon impact rather than simply depositing sediments, this introduces complexities to the parallelization layer of the method. Allowing

particles to split introduces unpredictability in the total number of particles that will exist in the simulation. This unpredictability can complicate the use of multi-threading. Future works includes finding a data structure allowing this splitting efficiently, leading to more realistic erosion patterns.

Simulation with multiple materials

One aspect we haven't addressed is a layered terrain with multiple materials. In the native way our method is done, we do not consider the transport of different materials (all sediments are considered as sand), but by storing a list of the different materials and the quantity transported by each particle, the same simulation process could be done at the cost of some memory and performance overhead.
Another possible adaptation of the erosion strategy for material voxels is to extend the erosion computation from binary voxels by define transformation rules from one material to another when a voxel is eroded a number $\#hits < -C$ or $\#hits > C$. For example, the material "clay" may transform to "sand" when eroded or to "rock" when many depositions occurred.

6.8 Conclusion

We introduced a flexible particle-based erosion system that is easy to use and simple to implement. We have presented how to adapt the process for various terrain representations and generate a variety of erosion phenomenon due to rain, wind, water bodies... by adjusting intuitive parameters hence generate automatically realistic 2.5D and 3D terrains. The use of external velocity fields provides a high flexibility i.e. using the simulations that best fits the user's needs (precision, control, implementation efficiency...). Our method can also be applied to underwater environments with identical physics simulation since our erosion method can be applied on 3D representations. Erosion algorithms are often limited to the use of height fields, but by finding more generalized methods, we can go toward a global use of 3D terrains, which can offer richer and more diverse landscapes.

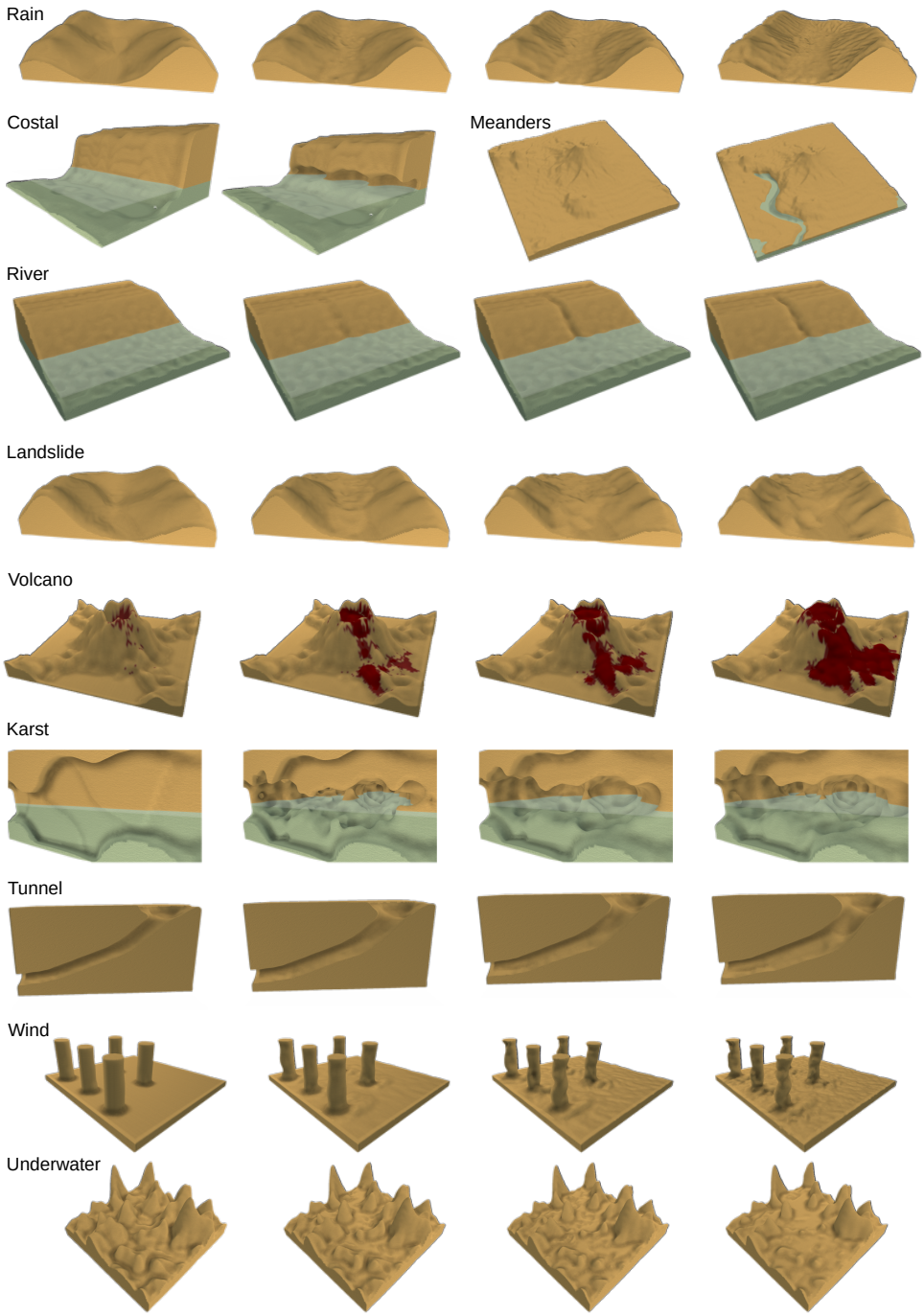


Figure 6.14: Erosion processes results on various representations presented in section 6.5. Used parameters used are detailed in Table 6.1.

Part IV

Conclusion

- ...

References

Abela, R., Liapis, A., & Yannakakis, G. N. (2015). A constructive approach for the generation of underwater environments. *Proceedings of the FDG workshop on Procedural Content Generation in Games* (cit. on p. 34).

Amawy, M. E., & Muftah, A. M. (2009). Karst development and structural relationship in the tertiary rocks of the al jabal al akhdar, ne libya: A case study in qasr libya area. *3rd International Symposium Karst Evolution in the South Mediterranean Area*, *14*, 173–189 (cit. on p. 53).

Argudo, O., Galin, E., Peytavie, A., Paris, A., & Guérin, E. (2020). Simulation, modeling and authoring of glaciers. *ACM Transactions on Graphics*, *39*, 1–14. https://doi.org/10.1145/3414685.3417855 (cit. on pp. 87, 91).

Baraff, D., & Witkin, A. (1998). Large steps in cloth simulation. *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998*, 43–54. https://doi.org/10.1145/280814.280821 (cit. on p. 96).

Beardall, M., Butler, J., Farley, M., & Jones, M. D. (2010). Directable weathering of concave rock using curvature estimation. *IEEE Transactions on Visualization and Computer Graphics*, *16*, 81–94. https://doi.org/10.1109/TVCG.2009.39 (cit. on pp. 101, 108).

Becher, M., Krone, M., Reina, G., & Ertl, T. (2017). Feature-based volumetric terrain generation. *Proceedings - I3D 2017: 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. https://doi.org/10.1145/3023368.3023383 (cit. on p. 99).

Beneš, B., & Forsbach, R. (2001). Layered data representation for visual simulation of terrain erosion. *Proceedings - Spring Conference on Computer Graphics, SCCG 2001*, 80–86. https://doi.org/10.1109/SCCG.2001.945341 (cit. on pp. 89, 90, 98).

Beneš, B., Těšínský, V., Hornyš, J., & Bhatia, S. K. (2006). Hydraulic erosion. *Computer Animation and Virtual Worlds*, *17*, 99–108. https://doi.org/10.1002/cav.77 (cit. on p. 90).

Bosscher, H., & Schlager, W. (1992). Computer simulation of reef growth. *Sedimentology*, *39*, 503–512. https://doi.org/10.1111/j.1365-3091.1992.tb02130.x (cit. on p. 34).

Brosz, J., Samavati, F. F., & Sousa, M. C. (2007). Terrain synthesis by-example. *Communications in Computer and Information Science*, *4 CCIS*, 58–77. https://doi.org/10.1007/978-3-540-75274-5_4 (cit. on p. 34).

Caretto, L. S., Gosman, A. D., Patankar, S. V., & Spalding, D. B. (1973). Two calculation procedures for steady, three-dimensional flows with recirculation. *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*, *19*, 60–68 (cit. on p. 105).

Collon, P., Bernasconi, D., Vuilleumier, C., & Renard, P. (2017). Statistical metrics for the characterization of karst network geometry and topology. *Geomorphology*, *283*, 122–142. https://doi.org/10.1016/j.geomorph.2017.01.034 (cit. on p. 78).

Collon, P., Steckiewicz-Laurent, W., Pellerin, J., Laurent, G., Caumon, G., Reichart, G., & Vaute, L. (2015). 3d geomodelling combining implicit surfaces and voronoi-based remeshing: A case study in the lorraine coal basin (france). *Computers and Geosciences*, *77*, 29–43. https://doi.org/10.1016/j.cageo.2015.01.009 (cit. on p. 78).

Cordonnier, G., Braun, J., Cani, M.-P., Benes, B., Galin, É., Peytavie, A., & Guérin, É. (2016). Large scale terrain generation from tectonic uplift and fluvial erosion. *Computer Graphics Forum*, *35*, 165–175. https://doi.org/10.1111/cgf.12820 (cit. on pp. 70, 91).

Cordonnier, G., Cani, M.-P., Beneš, B., Braun, J., & Galin, É. (2017a). Sculpting mountains: Interactive terrain modeling based on subsurface geology. *IEEE Transactions on Visualization and Computer Graphics*, *24*. https://doi.org/10.1109/TVCG.2017.2689022 (cit. on pp. 70, 87, 91, 109).

Cordonnier, G., Ecormier-nocca, P., Galin, É., Gain, J., Beneš, B., & Cani, M.-P. (2018). Interactive generation of time-evolving, snow-covered landscapes with avalanches. *Computer Graphics Forum*, *37*, 497–509. https://doi.org/10.1111/cgf.13379 (cit. on pp. 87, 91).

Cordonnier, G., Galin, É., Gain, J., Beneš, B., Guérin, É., Peytavie, A., & Cani, M.-P. (2017b). Authoring landscapes by combining ecosystem and terrain erosion simulation. *ACM Transactions on Graphics*, *36*. https://doi.org/10.1145/3072959.3073667 (cit. on pp. 87, 91).

Cordonnier, G., Jouvet, G., Peytavie, A., Braun, J., Cani, M.-P., Benes, B., Galin, E., Guérin, E., & Gain, J. (2023). Forming terrains by glacial erosion. *ACM*

Transactions on Graphics, *42*, 14. https://doi.org/10.1145/3592422\"{\i} (cit. on pp. 34, 87).

Cortial, Y., Peytavie, A., Galin, É., & Guérin, É. (2019). Procedural tectonic planets. *Eurographics Computer Graphics Forum*, *38*, 1–11. https://doi.org/10.1111/cgf.13614 (cit. on p. 34).

Cowart, L., Walsh, J. P., & Corbett, D. R. (2010). Analyzing estuarine shoreline change: A case study of cedar island, north carolina. *Journal of Coastal Research*, *265*, 817–830. https://doi.org/10.2112/jcoastres-d-09-00117.1 (cit. on p. 49).

Dey, R., Doig, J. G., & Gatzidis, C. (2018). Procedural feature generation for volumetric terrains using voxel grammars. *Entertainment Computing*, *27*, 128–136. https://doi.org/10.1016/j.entcom.2018.04.003 (cit. on p. 89).

Domínguez, L., Anfuso, G., & Gracia, F. J. (2005). Vulnerability assessment of a retreating coast in sw spain. *Environmental Geology*, *47*, 1037–1044. https://doi.org/10.1007/s00254-005-1235-0 (cit. on p. 49).

Droxler, A. W., & Jorry, S. J. (2021). The origin of modern atolls : Challenging darwin's deeply ingrained theory. *Annual Review of Marine Science*. https://doi.org/https://doi.org/10.1146/annurev-marine-122414-034137 (cit. on p. 69).

Ecormier-Nocca, P., Cordonnier, G., Carrez, P., Moigne, A. M., Memari, P., Benes, B., & Cani, M. P. (2021). Authoring consistent landscapes with flora and fauna. *ACM Transactions on Graphics*, *40*. https://doi.org/10.1145/3450626.3459952 (cit. on p. 40).

Eisemann, E., & Decoret, X. (2008). Single-pass gpu solid voxelization for real-time applications. *Proceedings - Graphics Interface*, 73–80 (cit. on p. 89).

Emilien, A., Poulin, P., Cani, M.-P., & Vimont, U. (2015). Interactive procedural modelling of coherent waterfall scenes. *Computer Graphics Forum*, *34*, 22–35. https://doi.org/10.1111/cgf.12515 (cit. on p. 89).

Fernandes, G. D., & Fernandes, A. R. (2018). Space colonisation for procedural road generation. *2018 International Conference on Graphics and Interaction (ICGI)*, 1–8. https://doi.org/10.1109/ITCGI.2018.8602928 (cit. on p. 78).

Gain, J., Marais, P., & Straßer, W. (2009). Terrain sketching. *Proceedings of I3D 2009: The 2009 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, *1*, 31–38. https://doi.org/10.1145/1507149.1507155 (cit. on pp. 34, 70, 89).

Galin, E., Guérin, E., Peytavie, A., Cordonnier, G., Cani, M.-P., Benes, B., & Gain, J. (2019). A review of digital terrain modeling. *Computer Graphics Forum*, *38*, 553–577. https://doi.org/10.1111/cgf.13657 (cit. on pp. 3, 34, 86, 89).

Galin, É., Peytavie, A., Maréchal, N., & Guérin, É. (2010). Procedural generation of roads. *Computer Graphics Forum*, *29*, 429–438. https://doi.org/10.1111/j.1467-8659.2009.01612.x (cit. on p. 78).

Goes, F. D., & James, D. L. (2017). Regularized kelvinlets: Sculpting brushes based on fundamental solutions of elasticity. *ACM Transactions on Graphics*, *36*, 401–411. https://doi.org/10.1145/3072959.3073595 (cit. on pp. 39, 46).

Grosbellet, F., Peytavie, A., Guérin, É., Galin, É., Mérillou, S., & Benes, B. (2016). Environmental objects for authoring procedural scenes. *Computer Graphics Forum*, *35*, 296–308. https://doi.org/10.1111/cgf.12726 (cit. on p. 35).

Guérin, E., Peytavie, A., Masnou, S., Digne, J., Sauvage, B., Gain, J., & Galin, E. (2022). Gradient terrain authoring. *Computer Graphics Forum*, *41*, 85–95. https://doi.org/10.1111/cgf.14460 (cit. on pp. 90, 99).

Guérin, É., Digne, J., Galin, É., & Peytavie, A. (2016). Sparse representation of terrains for procedural modeling. *Computer Graphics Forum*, *35*, 177–187. https://doi.org/10.1111/cgf.12821 (cit. on pp. 35, 90).

Guérin, É., Digne, J., Galin, É., Peytavie, A., Wolf, C., Beneš, B., & Martinez, B. (2017). Interactive example-based terrain authoring with conditional generative adversarial networks. *ACM Transactions on Graphics*, *36*. https://doi.org/10.1145/3130800.3130804 (cit. on p. 34).

Hong, Q. (2013). A skeleton-based technique for modelling implicit surfaces. *Proceedings of the 2013 6th International Congress on Image and Signal Processing, CISP 2013*, *2*, 686–691. https://doi.org/10.1109/CISP.2013.6745253 (cit. on p. 110).

Huang, Z., Nichol, S. L., Harris, P. T., & Caley, M. J. (2014). Classification of submarine canyons of the australian continental margin. *Marine Geology*, *357*, 362–383. https://doi.org/10.1016/j.margeo.2014.07.007 (cit. on p. 53).

Ito, T., Fujimoto, T., Muraoka, K., & Chiba, N. (2003). Modeling rocky scenery taking into account joints. *Proceedings of Computer Graphics International Conference, CGI, 2003-Janua*, 244–247. https://doi.org/10.1109/CGI.2003.1214475 (cit. on p. 89).

Jones, B. D., & Williams, J. R. (2017). Fast computation of accurate sphere-cube intersection volume. *Engineering Computations*, *34*, 1204–1216. https://doi.org/10.1108/EC-02-2016-0052 (cit. on p. 98).

Kapp, K., Gain, J., Guérin, E., Galin, E., & Peytavie, A. (2020). Data-driven authoring of large-scale ecosystems. *ACM Transactions on Graphics*, *39*. https://doi.org/10.1145/3414685.3417848 (cit. on p. 34).

Kass, M., Witkin, A., & Terzopoulos, D. (1988). Snakes: Active contour models. (Cit. on p. 43).

Kaufman, A., Cohen, D., & Yagel, R. (1993). Volume graphics. *Computer*, *26*, 51–64. https://doi.org/10.1109/MC.1993.274942 (cit. on p. 89).

Koschier, D., Bender, J., Solenthaler, B., & Teschner, M. (2022). A survey on sph methods in computer graphics. *Computer Graphics Forum*, *41*, 737–760. https://doi.org/10.1111/cgf.14508 (cit. on p. 95).

Krištof, P., Beneš, B., Křivánek, J., & Šťava, O. (2009). Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum*, *28*, 219–228. https://doi.org/10.1111/j.1467-8659.2009.01361.x (cit. on pp. 88, 90, 92, 95, 103).

Lengyel, E. (2010). Voxel-based terrain for real-time virtual simulations, 148 (cit. on p. 89).

Li, W. (2021). Procedural modeling of the great barrier reef. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *13017 LNCS*, 381–391. https://doi.org/10.1007/978-3-030-90439-5_30 (cit. on p. 34).

Mareschal, J. C. (1989). Fractal reconstruction of sea-floor topography. *Pure and Applied Geophysics PAGEOPH*, *131*, 197–210. https://doi.org/10.1007/BF00874487 (cit. on p. 34).

Mei, X., Decaudin, P., & Hu, B. G. (2007). Fast hydraulic erosion simulation and visualization on gpu. *Proceedings - Pacific Conference on Computer Graphics and Applications*, 47–56. https://doi.org/10.1109/PG.2007.27 (cit. on pp. 90, 108, 109).

Michel, E., Emilien, A., & Cani, M.-P. (2015). Generation of folded terrains from simple vector maps. *Eurographics 2015 short paper proceedings*, 4–8. https://doi.org/10.2312/egsh.20151019 (cit. on p. 34).

Musgrave, F. K., Kolb, C. E., & Mace, R. S. (1989). The synthesis and rendering of eroded fractal terrains. *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1989*, 41–50. https://doi.org/10.1145/74333.74337 (cit. on pp. 34, 86, 90).

Neidhold, B., Wacker, M., & Deussen, O. (2005). Interactive physically based fluid and erosion simulation. *Natural Phenomena*, 25–32 (cit. on pp. 86, 90).

O'Brien, J. F., & Hodgins, J. K. (1995). Dynamic simulation of splashing fluids. *Proceedings Computer Animation, CA 1995*, 198–205. https://doi.org/10.1109/CA.1995.393532 (cit. on p. 108).

Olsen, J. (2004). Realtime procedural terrain generation. *Department of Mathematics And Computer Science (. . .*, 20 (cit. on p. 86).

Onoue, K., & Nishita, T. (2000). A method for modeling and rendering dunes with wind-ripples. *Proceedings - Pacific Conference on Computer Graphics and Applications*, *2000-Janua*, 427–428. https://doi.org/10.1109/PCCGA.2000.883978 (cit. on p. 109).

Oron, S., Akkaynak, D., Tchernov, B. N. G., & Shaked, Y. (2023). How monster storms shape fringing reefs: Observations from the 2020 middle east cyclone. *Ecosphere*, *14*. https://doi.org/10.1002/ecs2.4602 (cit. on p. 49).

Paris, A., Guérin, E., Peytavie, A., Collon, P., & Galin, E. (2021). Synthesizing geologically coherent cave networks. *Computer Graphics Forum*, *40*, 277–287. https://doi.org/10.1111/cgf.14420 (cit. on pp. 77, 79, 104).

Paris, A., Peytavie, A., Guérin, E., Argudo, O., & Galin, E. (2019a). Desertscape simulation. *Computer Graphics Forum*, *38*, 47–55. https://doi.org/10.1111/cgf.13815 (cit. on pp. 46, 90, 105, 109).

Paris, A., Galin, E., Peytavie, A., Guérin, E., & Gain, J. (2019b). Terrain amplification with implicit 3d features. *ACM Transactions on Graphics*, *38*, 1–15. https://doi.org/10.1145/3342765 (cit. on pp. 34, 90, 102, 107).

Patel, D., Natali, M., Lidal, E. M., Parulek, J., Brazil, E. V., & Viola, I. (2021). Modeling terrains and subsurface geology. *Interactive Data Processing and 3D Visualization of the Solid Earth*, 1–43. https://doi.org/10.1007/978-3-030-90716-7_1 (cit. on pp. 34, 70).

Peytavie, A., Galin, E., Grosjean, J., & Merillou, S. (2009). Arches: A framework for modeling complex terrains. *Computer Graphics Forum*, *28*, 457–467. https://doi.org/10.1111/j.1467-8659.2009.01385.x (cit. on pp. 89, 90, 98).

Pratt, M. J., Wysession, M. E., Aleqabi, G., Wiens, D. A., Nyblade, A. A., Shore, P., Rambolamanana, G., Andriampenomanana, F., Rakotondraibe, T., Tucker, R. D., Barruol, G., & Rindraharisaona, E. (2017). Shear velocity structure of the crust and upper mantle of madagascar derived from surface wave tomography. *Earth and Planetary Science Letters*, *458*, 405–417. https://doi.org/10.1016/j.epsl.2016.10.041 (cit. on p. 53).

Pytel, A., & Mann, S. (2015). Procedural modeling of cave-like channels. (Cit. on p. 77).

Ranz, W. E., Talandis, G. R., & Gutterman, B. (1960). Mechanics of particle bounce. *AIChE Journal*, *6*, 124–127. https://doi.org/10.1002/aic.690060123 (cit. on p. 111).

Richardson, J. F., & Zaki, W. N. (1954). The sedimentation of a suspension of uniform spheres under conditions of viscous flow. *Chemical Engineering Science*, *3* (cit. on p. 94).

Rigaudière, D., Gesquière, G., & Faudot, D. (2000). Shape modelling with skeleton based implicit primitives. *Methods* (cit. on p. 110).

Roa, T., & Benes, B. (2004). Simulating desert scenery. *Winter School of Computer Graphics SHORT communication Papers Proceedings*, 17–22 (cit. on pp. 90, 109).

Roose, D., Leuven, K. U., & López, Y. R. (2011). Dynamic refinement for fluid flow simulations with sph particle refinement for fluid flow simulations with sph. (Cit. on p. 95).

Roudier, P. (1993). Synthèse de paysages réalistes par simulation de processus d'érosion (cit. on p. 86).

Runions, A. (2008). Modeling biological patterns using the space colonization algorithm. *A Thesis submitted to the faculty of graduate studies for degree of master of science*, 74 (cit. on p. 78).

Schott, H., Paris, A., Fournier, L., Guérin, E., & Galin, E. (2023). Large-scale terrain authoring through interactive erosion simulation. *ACM Transactions on Graphics*, *42*, 1–15. https://doi.org/10.1145/3592787 (cit. on p. 34).

Shadrick, J. R., Rood, D. H., Hurst, M. D., Piggott, M. D., Hebditch, B. G., Seal, A. J., & Wilcken, K. M. (2022). Sea-level rise will likely accelerate rock coast cliff retreat rates. *Nature Communications*, *13*. https://doi.org/10.1038/s41467-022-34386-3 (cit. on p. 53).

Smelik, R. M., Kraker, K. J. D., Groenewegen, S. A., Tutenel, T., & Bidarra, R. (2009). A survey of procedural methods for terrain modelling. *Proceedings of the CASA workshop on 3D advanced media in gaming and simulation (3AMIGAS)* (cit. on p. 86).

Stachniak, S., & Stuerzlinger, W. (2005). An algorithm for automated fractal terrain deformation. *In Proceedings of Computer Graphics and Artificial Intelligence*, 64–76 (cit. on p. 86).

Stam, J. (1999). Stable fluids. *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1999*, 121–128. https://doi.org/10.1145/311535.311548 (cit. on p. 105).

Stam, J. (2003). Flows on surfaces of arbitrary topology. *ACM Transactions on Graphics*, *22*, 724–731. https://doi.org/10.1145/882262.882338 (cit. on p. 102).

Stokes, G. G. (1850). On the effect of the internal friction of fluids on the motion of pendulums. Pitt Press Cambridge. https://doi.org/10.1017/CBO9780511702266.002 (cit. on p. 94).

Swope, W. C., Andersen, H. C., Berens, P. H., & Wilson, K. R. (1982). A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal of Chemical Physics*, *76*, 637–649. https://doi.org/10.1063/1.442716 (cit. on p. 96).

Talgorn, F. X., & Belhadj, F. (2018). Real-time sketch-based terrain generation. *ACM International Conference Proceeding Series*, 13–18. https://doi.org/10.1145/3208159.3208184 (cit. on p. 34).

Tychonievich, L. A., & Jones, M. D. (2010). Delaunay deformable mesh for the weathering and erosion of 3d terrain. *Visual Computer*, *26*, 1485–1495. https://doi.org/10.1007/s00371-010-0506-2 (cit. on p. 95).

Verlet, L. (1967). Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical Review*, *159*, 98–103. https://doi.org/10.1103/PhysRev.159.98 (cit. on p. 96).

Vila-Concejo, A., & Kench, P. (2016, August). Storms in coral reefs. Wiley Blackwell. https://doi.org/10.1002/9781118937099.ch7 (cit. on p. 49).

Wejchert, J., & Haumann, D. (1991). Animation aerodynamics. *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1991*, *25*, 19–22. https://doi.org/10.1145/122718.122719 (cit. on pp. 35, 39, 46).

Wojtan, C., Carlson, M., Mucha, P. J., & Turk, G. (2007). Animating corrosion and erosion. *Natural Phenomena*, 15–22 (cit. on pp. 92–95).

Yan, P., Zhang, J., Kong, X., & Fang, Q. (2020). Numerical simulation of rockfall trajectory with consideration of arbitrary shapes of falling rocks and terrain. *Computers and Geotechnics*, *122*. https://doi.org/10.1016/j.compgeo.2020.103511 (cit. on pp. 94, 111).