THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En informatique

École doctorale I2S

Unité de recherche LIRMM

Génération procédurale d'environnements sous-marins
Procedural generation of underwater environments

Présentée par Marc HARTLEY
le [XX mois année]

Sous la direction de Christophe FIORIO, Noura FARAJ et Karen GODARY-DEJEAN

Devant le jury composé de

[Prénom NOM, Titre, Affiliation]	[Statut jury]
[Prénom NOM, Titre, Affiliation]	[Statut jury]
[Prénom NOM, Titre, Affiliation]	[Statut jury]
[Prénom NOM, Titre, Affiliation]	[Statut jury]
[Prénom NOM, Titre, Affiliation]	[Statut jury]
[Prénom NOM, Titre, Affiliation]	[Statut jury]
[Prénom NOM, Titre, Affiliation]	[Statut jury]

UNIVERSITÉ DE MONTPELLIER

Contents

Abstract

This thesis, entitled *"Procedural terrain generation for underwater environments"*,explores the specialized area of procedural terrain generation, specifically targeting underwater settings. Procedural terrain generation remains a vibrant research area within computer graphics, particularly as advancements in simulation, rendering, and interaction techniques have fostered increased collaboration with terrain experts across various disciplines.

Virtualizing the physical world enables users to observe and interact with it in ways that enhance understanding and break down the boundaries between scientific fields. Terrain science, by its nature, brings together a diverse range of experts, including geologists, oceanologists, physicists, meteorologists, biologists, roboticists, computer scientists, and 3D artists. This thesis focuses on involving users in the creation of virtual worlds through fast and controllable algorithms, with a particular emphasis on underwater environments.

The thesis is structured into three parts, guiding the reader from the initial design of a landscape through to its 3D modeling and final refinement.

In the first part, we introduce a formal approach to terrain design, allowing environments to be conceived in semantic terms, abstracting away from the complexities of 3D geometry and data structures.

The second part focuses on translating these conceptual environments into 3D form. We present new models for generating and modeling coral reef islands and karst networks.

Finally, in the third part, we concentrate on enhancing the realism of these 3D terrains through physical simulations of erosion processes. We demonstrate a flexible and controllable method for simulating 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 needs. As these objectives get progressively achieved, a new trend has started to emmerge in which focus is pushed toward user control. The focus of this work is on one specific branch of terrain generation: the interactive creation and modeling of landscapes.

The utility of landscape generation goes beyond purely scientific pursuits, finding applications in diverse fields such as biology, geology, and robotics. Additionally, it has become a central tool in the entertainment industry, particularly in video games and cinema, where creating realistic and dynamic environments is key for immersive experiences. The ability to generate landscapes that help in understanding natural rules and testing hypotheses makes this the ideal tool for both researchers and industries.

What sets this work apart is its emphasis on underwater landscapes or "seascapes", an area still overviewed but is important in domains such as 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.

However, this exploration of the domain comes with significant difficulties. Finding a balance between automation and the user's creative desires is essential, as is the ability to isolate variables and manage scaling effectively. 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 representations, fluid solvers, landscape types, user's hardwares, or objectives, 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 replacing the users themselves.

1.1 Contributions and outlines

This thesis proposes several contributions in different branches of terrain generation. We will divide these works in three parts: the semantic representation of environments, the modeling of specific features and the terrain enhancement. Those three parts are presented in the chronological order in which a modeler would create his work: first drawing the esquisse of his artwork to see what it could look like coarsely at the end, then produce each element one by one, and finally add the final details that improve the whole.

This research follows the same order for terrain generation, beginning from the development of an abstract representation that bridges the gap between computer science and Earth science, offering a generalization of the desired landscape type, with a particular focus on underwater environments, but also applicable to terrestrial landscapes (Chapter 3). Secondly, we will develop on the problematic of 3D modeling of some natural features, and more specifically on coral reef islands (Chapter 2) and karst networks (Chapter D). Finally, we propose an erosion method that amplify the realism of the virtual terrain (Chapter 5).

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.

Semantics

The semantic representation we present in the first part of this work aims to design the features of a terrain with an abstraction the 3D aspect of the surface. We will see in the Chapter 3 the introduction of environmental objects and their simplified representation, used to have a symbolic representation of the terrain features, biotic and abiotic, that are present in the scene. Using symbolism allows us to focus on the interactions between the different elements of an environment without the high computational needs of running an accurate multiphysics simulation. Moreover, the simplified representation used allows the user to manipulate the shape of the final terrain, without having to choose a specific terrain representation.

Modeling

As the global layout of the terrain is defined, the 3D modeling of the terrain surface can begin. Many methods have been proposed in the computer graphics community to shape specific terrain features using procedural methods. We will propose two new contributions to these methods.

In Chapter 2, we present a method to model a coral reef island from a sketch by using the properties that have been observed in travel journals, such as the radiality, the global structure of the visual features, the typical profile. The method is simple but constraint. We then use the procedural paradigm of sketching to create a large synthetic dataset of terrains, that is fed in a deep learning generative model to unlock the limitations of the initial sketching. This pipeline shows that we can first use the given properties to create a simplified model and then, by simply fine-tuning an off-the-shelf pretrained GAN model, remove these constraints to improve the user's possibilities.

In Chapter D, we propose a new algorithm for the generation of karst networks. This geomorphological feature has been studied previously with a geologist point of view. We provide a new viewpoint on the problem by focusing on the non-expert user control to shape the structure of the network by using a directed acyclic graph, which closely resembles a tree structure. This method is enhanced by a fractal generation approach with cycles, allowing for iterative generation of complex landscapes.

Enhancement

To increase the realism and the visual impact of the generated synthetic landscapes, the use of terrain enhancement techniques are often required. We tackled the specific challenge of running erosion simulations, a type of enhancement that mimics the effects of water, wind and erosive forces on a virtual terrain to improve the belivability of the final landscape. A particle-based erosion method is proposed in Chapter 5, designed to be generalizable for flexibility on multiple levels, and its implementation is oriented towards speed and parallelization. The main flexibility of our method is to be applicable to many terrain representations, but also to be agnostic to the fluid solver used and to be generalized for both landscapes and seascapes.

Due to the disparities of works for the different topics proposed in this thesis, we will introduce the topic, present a state of the art and develop our methods in each chapter.

1.2 Procedural generation

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 musics, textures, or virtual ecosytem.

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.

One of the main benefits of procedural generation is its ability to generate content with minimal storage requirements. Content is typically generated in real-time or on-the-fly, rather than being explicitly stored, which allows for the creation of large amount of content without requiring significant storage space. Additionally, 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 widely used across various fields to create dynamic, diverse, and detailed content efficiently. In video games, it enables the automatic creation of expansive game worlds, complex level designs, and varied character animations. For example, procedural generation can produce entire game levels with unique layouts for each playthrough, enhancing replayability, or generate diverse character behaviors that adapt to in-game environments, making interactions more dynamic and lifelike.

Beyond video games, procedural generation plays a significant role in other industries as well. In the film industry, it is used to generate intricate visual effects, such as realistic smoke, fire, and water simulations, which would be labor-intensive to create manually (Stam, 2003b; Hädrich et al., 2021; Fedkiw and Jensen, 2001; Xing et al., 2016). Facial animation also benefits from procedural techniques, where algorithms can generate subtle, lifelike expressions that enhance the realism of CGI characters [ADD REFERENCES]. Crowd simulations are another critical application, with procedural methods used to populate scenes with thousands of unique, animated characters, each with distinct appearances and behaviors (Yersin et al., 2009; Park, 2010; Narain, 2011).

In simulations, procedural generation is integral to creating accurate and scalable representations of real-world environments. This includes the generation of complex ecosystems for environmental simulations, detailed urban layouts for city planning, and even diverse astronomical phenomena for space simulations [ADD REFERENCES]. Additionally, procedu-

ral generation allows researchers to isolate and control variables within these simulations, enabling them 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. Mathematical models and noise functions, such as Perlin or Simplex noise, are often used to create the foundational structures in procedural content. These algorithms can generate everything from the random yet cohesive textures of terrain to the complex, non-repetitive patterns seen in natural phenomena or digital art (Vergne et al., 2011). This might involve generating natural features like landscapes, complex textures and mesostructures on textile objects (Michel and Boubekeur, 2023), or even entire ecosystems. Incorporating elements of real-world phenomena, such as erosion patterns on the objects geometry or texture, like cracks, bumps or melting, make them more believable and diverse (Gobron and Chiba, 2001; Wojtan et al., 2007; Iwasaki et al., 2010).

The flexibility of these algorithms not only allows for the creation of complex content but also enables user-driven customization that 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. A urban layout can, for example, be constraint to certain areas (Zhou et al., 2021), to satisfy inhabitants accessibility needs (Lima et al., 2022), to minimize energy consumption (Shi et al., 2017) or even solar irradiations (Vermeulen et al., 2015).

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. Such predictability is useful in scenarios where consistency and reliability are important. For example, in engineering simulations, deterministic systems ensure that specific results can be consistently reproduced, which is essential for validation and testing. Similarly, in video game development, deterministic algorithms might be employed to generate identical game worlds across different sessions, ensuring that all players experience the same environment under the same conditions, and that no unexpected obstacle blocks the character. This consistency is also beneficial in content creation pipelines, where artists or designers need to fine-tune specific aspects of generated content. Since the output is repeatable, they can iteratively adjust parameters and observe how these changes affect the final result, making it easier to achieve the desired outcome.

On the other hand, stochastic systems introduce elements of randomness into the generation process, resulting in varied outputs even when the same initial parameters are used. This randomness is particularly important in applications where variation and uniqueness are desired. In video games, for instance, stochastic systems might be utilized to create a wide variety of levels, landscapes, or in-game items, ensuring that each playthrough feels different and offers a new experience. Beyond that, stochastic processes are also used in fields like procedural music generation, where each composition can be unique, providing a unique listening experience every time.

Recognizing the strengths and limitations of both deterministic and stochastic systems, many applications integrate these approaches into hybrid systems. In such systems, deterministic algorithms might be used to establish the overall structure, ensuring consistency and adherence to specific rules or constraints. Then stochastic elements are introduced to fill in details and add variability, creating dynamic content that remains coherent. For example, in

procedural architecture, a deterministic system might generate the building's foundational layout, while stochastic processes add varied interior details, making each design unique yet structurally coherent. This hybrid approach can provide a balance between consistency and variability, making it possible to create content that is both reliable and dynamic.

In rule-based systems, content is generated by following a set of predefined rules, which can range from simple conditions to complex algorithms. These rules dictate how different elements interact, ensuring that the output adheres to specific constraints. For instance, in procedural city generation, rules might govern the placement of buildings, roads, and parks to create a realistic urban layout. In this way, rule-based systems ensure that the generated content is both structured and coherent. Other examples of rule-based systems, in game design, might be used to generate levels that adhere to specific gameplay mechanics, such as ensuring that obstacles are placed in a way that challenges the player without making the game impossible. In architecture, rules might dictate the structural integrity of a building, ensuring that generated designs are not only aesthetically pleasing but also feasible.

(Temuçin et al., 2020)

Typically, rule-based systems involves iterative processes, where initial outputs are continuously refined or adjusted based on additional rules or feedback. This iterative approach allows for the progressive enhancement of the content, where each cycle of generation can introduce new elements or improve upon previous iterations, resulting in a more polished and coherent final output. Those systems are mainly considered deterministic but including a slight randomness may increase the variability and provide a more natural feeling.

Procedural generation significantly help reducing manual efforts, which becomes unavoidable for the creation of large assets or a diversity of variations of content. Including control over the algorithms in play allows the user to adapt the results to its needs while minimizing the required storage.

1.2.1 Notable examples in the 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 (*Rogue*, 1980). 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 (*Elite*, 1984). 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 (*Dwarf Fortress*, 2006). 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 (*Spore*, 2008). 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 (*Minecraft*, 2011). 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 modeling natural disaster (Valette et al., 2005; Hudák and Ďurikovič, 2011; Hädrich et al., 2021), animal behaviours (Ecormier-Nocca et al., 2021; Wampler and Popovi, 2009), robot simulations (Louis et al., 2019; Lejeune, 2021), military simulations (Dam et al., 2019), etc... 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 (Gobron and Chiba, 2001), 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.

1.2.2 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 Procedural terrain generation

1.3.1 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 (Galin, Guérin, et al., 2019).

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.

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. (de Araújo et al., 2015)

Layered models

Layered models are a type of volumetric representation that encode different material layers within the terrain and are defined by a function $\mu : \mathbb{R}^3 \mapsto \mathcal{M}$, where $\mathcal{M}$ 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) (Laine and Karras, 2010) or voxel Directed Acyclic Graphs (DAG) (Villanueva et al., 2017; Careil et al., 2020). 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 $\mu : \mathbb{R}^3 \mapsto \mathcal{M}$, 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.

1.3.2 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 (Cattaneo and Jocher, 2005; Boldea, 2009; Menshutina et al., 2020) to vegetation patterns (Greene, 1989).

Large-scale terrain generation involves creating expansive terrains that exhibit natural-looking landscapes over large areas. The main method for generating a large-scale terrain is the employ of the noise functions, but other techniques like subdivision schemes 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 often used for generating the broad, natural features characteristic of large-scale terrains, although they may lack the precision needed for detailed landform creation. Large-scale methods are mostly defined in 2.5D. Gravity playing a large role in the shape of such environment, the surface of the terrain contains few concavities. Such cases, like caves, arches or overhangs, are mainly considered in the feature-scale.

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

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, Jouvet, et al., 2023; Schott et al., 2023; Paris, Galin, et al., 2019). This process can be expensive in time but results

in more plausible surfaces.

Procedural landform generation can be focused 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.

1.3.3 User interaction [TODO]

Procedural terrain generation algorithms usually take parameters into account in order to generate something that fits the user's needs. The tools that let the user interact with the different parameters are essential to include efficiently the user in the process.

The parameters that a user can play with can have many nature, and require a large varety of visual tools to interact with them. In the different nature of parameters, we can typically find:

Noise parameters: Almost abstract values used in the equation ruling the noise functions.

Physical constants: Constants that are used in physic simulations. They are often tweaked by the user to exagerate certain features, force some phenomena, etc... As they

are constants, the parametrization of these values is often done inside the source code, as spinboxes, or inside a configuration file.

Densities and distributions: Marking areas that are affected by a process, or where a seed can be used is useful to control a procedural algorithm. This is refered as "masking", but due to its binary nature, we can find sharp and unatural transitions from the use of binary masks. We often use weighted masks to provide more control about probabilities that a seed appear at one point. The control of the masks is usually done through grayscale images or by a "brush tool" that can modify the value directly on the surface of a 2D or 3D object. As virtual terrains are generally represented by a rectangular base, the use of brushes is a facade to the manipulation of a grayscale image.

Structures placement: Placing a specific terrain feature inside a terrain may be useful to finalize the generation of a landscape, or to add obstacles that will affect a physic simulation. This is usually implemented by clicking on the surface of the terrain, to provide the initial *xyz* position, and then use translation and rotation tools to adjust the placement. If many elements are added, we usually use the previous strategy, involving masking, to add all the structures.

Manual manipulation of the terrain: In order to customize at a lower level the surface of the terrain, the user may be able to alter the surface of the terrain by using brushes. Clicking or dragging a brush on the surface usually result in the rise or lowering of the surface level around the brush. This process modifies the field representing the terrain. Procedural brushes can imply more complex behaviours, and may use the velocity of the brush stroke in its input.

Each developer uses different strategies for each type of parameters as the needs or objectives of each application is different. Providing the user with too many parameters can be overwhelming, while too few can limit the output possibilities. In the meantime, providing unintuitive parameters like dimensionless values often result in trial and error strategy, requiering to run the generation algorithm many times before finding the appropriate values. This implies that the algorithms must be able to be executed fast. On an opposite side, removing some user controls to use hard-coded constants instead can greatly increase the speed. Finally, an algorithm that aims to be fast or let the user many parameters to control may reduce the realism of the output. Most algorithms try to strike a balance between realism, speed and control.

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. Usually, a realistic algorithm tends to have low user control and speed.

On another hand, 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.

Offline: generation that takes longer, often used for precomputed content or offline rendering.

The execution speed of a procedural algorithm is crucial as the user will fine-tune the parameters before being satisfied. 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.

1.3.4 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.

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f} \tag{1.1}$$

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:

$$\nabla \cdot \mathbf{u} = 0 \tag{1.2}$$

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 (Stam, 1999). 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 (Brackbill et al., 1988). 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) (Müller et al., 2003) 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 (Koschier et al., 2022). 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.

We can find improvements (Roose et al., 2011), can represent may elements (Iwasaki et al., 2010), either fluid or sediments (Lenaerts and Dutré, 2009), so it is often used to simulate

accurately water bodies (Nikeghbali and Omidvar, 2018).

Some other models exist, and are often proposed in OpenFOAM, either using grids, or particles, or an hybrid version (Caretto et al., 1973) [ADD OTHER REFERENCES]. But some works tend in directions that are more "procedural" like the use of cellular automata (Boldea, 2009; Cattaneo and Jocher, 2005), while other toward a more modern approach with the use of deep learning (Tompson et al., 2017) [ADD OTHER REFERENCES].

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 reef 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. Geo-

logical 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.

Paris, *Modeling and simulating virtual terrains* (2023) 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)

Coral reefs have intrigued humans for centuries, with early observations by ancient civilizations often misunderstanding these structures as mysterious sea plants or rocks. The study of coral reefs gained scientific momentum during the era of exploration, particularly through the work of figures like Captain James Cook and Charles Darwin. Darwin's theory of atoll formation, developed during his voyage on the HMS Beagle, was a significant milestone,

proposing that atolls form around sinking volcanic islands as coral growth keeps pace with subsidence.

The 20th century brought technological advancements, such as scuba diving and sonar, which allowed scientists to directly observe and map coral reefs, deepening our understanding of their biology and ecological significance. In recent decades, tools like remote sensing and genetic analysis have further expanded our knowledge, highlighting the critical role of coral reefs in marine biodiversity and their vulnerability to environmental changes.

Anatomy of coral reefs

Coral polyps

Coral polyps are the essential building blocks of coral reefs. These tiny, soft-bodied organisms, just a few millimeters in diameter, form the vast structures of coral reefs. Each polyp consists of a sac-like body with a central mouth surrounded by tentacles equipped with stinging cells (nematocysts) for defense and capturing prey.

The body of the polyp is anchored to a hard surface, which it helps create by secreting calcium carbonate $CaCO_3$. This secretion forms a protective exoskeleton around the polyp, allowing it to attach securely to the substrate or to other polyps in a colony. This secretion process creates the coral's hard structure, which becomes the foundation of the reef.

Reproduction in coral polyps occurs in two main ways: asexual and sexual reproduction. Asexually, polyps reproduce through a process called budding, where new polyps form from the sides of an existing polyp. These new polyps remain attached, leading to the growth of a larger coral colony. In sexual reproduction, polyps release eggs and sperm into the water during spawning events, where fertilization occurs externally. The resulting larvae settle onto a suitable substrate, developing into new polyps and eventually forming new colonies.

Coral polyps are in a symbiotic relationship with zooxanthellae, a type of photosynthetic algae that live within the tissues of the coral polyps, providing them with vital energy through photosynthesis, fueling its growth and calcification.

In return, the coral provides the zooxanthellae with protection and access to sunlight. This mutualistic relationship is highly efficient, allowing corals to thrive in the nutrient-poor waters of tropical oceans.

However, this relationship is fragile. When corals are exposed to environmental stressors such as elevated water temperatures, they may expel their zooxanthellae in a phenomenon known as coral bleaching. Without the algae, the coral loses its primary energy source, leading to a whitening of the coral and, if conditions do not improve in the next weeks, eventual death.

Coral colonies

Coral polyps, though individually small and simple, have the remarkable ability to form large, complex colonies that create the structure of coral reefs. These colonies exhibit a variety of growth patterns, each of which plays a different role in the overall architecture and ecological function of coral reefs.

Once established, coral colonies can develop into a wide range of growth forms, each adapted to specific environmental conditions and with unique functions to contribute to the reef shape.

Massive corals, like those in the genus *Porites*, grow in boulder-like forms. They tend to grow more slowly than other types but are much more resistant to physical damage from waves and storms. They contribute to the long-term stability and resilience of coral reefs. Massive corals can live for hundreds of years, with some colonies growing to several meters

in diameter. Their dense, solid structure provides a stable and enduring foundation for the reef, making them useful for the long-term stability of the reef ecosystem. Their robust skeletons can withstand strong wave action, storms, and other environmental stresses that would easily damage more delicate coral types. As they grow and expand, massive corals create the core of the reef, providing a solid base that other corals and marine organisms can build upon. Their presence is vital for maintaining the reef's overall integrity and resilience.

In contrast, branching corals, such as those in the genus *Acropora* (e.g., staghorn corals), look like a tree structure with numerous branches. These corals grow rapidly, extending outward to maximize exposure to sunlight. The dense thickets formed by branching corals provide essential habitat for a wide variety of marine species, from small fish to invertebrates. This growth form is particularly present in shallow, well-lit waters where competition for space and light is intense. This rapid growth allows them to quickly occupy space and expand the reef's surface area. The three-dimensional architecture of branching corals provides important habitats for a variety of marine life, including small fish and invertebrates, offering them protection from predators and strong currents. However, branching corals are relatively fragile and can be easily damaged by storms, waves, or human activity. Their ability to regenerate through fragmentation allows them to recover quickly from physical damage, but they do not contribute significantly to the reef's overall structural strength.

Similarly, foliose corals like in the genus *Astreopora*, which form leaf-like structures, add another dimension to reef complexity. These corals grow in overlapping layers, creating sheltered environments within the reef that are ideal for smaller marine organisms. Foliose corals can be particularly abundant in areas with moderate water movement, where their shape helps capture food particles from the water.

Corals that grow in flat, horizontal formations are known as table corals or plate coral, such as *Acropora*, can spread out to cover large areas, creating expansive, flat surfaces that provide important habitats for animals and other coral colonies. Their growth form allows them to efficiently capture sunlight, making them well-suited for shallow reef environments.

Some corals, known as encrusting corals like in the genus *Cyphastrea*, grow as a thin layer over hard substrates, including rocks, dead coral skeletons, and other surfaces. They are particularly adept at covering large areas of the reef, often outcompeting other organisms for space. Their growth form helps to bind loose sediments and rubble together, contributing to the stabilization of the reef's foundation. Although encrusting corals do not add significant height or volume to the reef, they are critical for cementing the reef structure, preventing erosion, and enhancing the overall stability of the reef. By filling in gaps between other coral colonies and consolidating the reef framework, encrusting corals help maintain the reef's structural cohesion, which is essential for its longevity.

The position of different coral types within the reef varies depending on factors such as water depth, wave exposure, and light availability. Massive corals are typically found on the reef slope, where the water is deeper and more stable. Encrusting corals thrive in the less turbulent waters of the deep reef, where they can spread across stable surfaces. Branching corals, such as Acropora species, are often found on the reef crest and fore reef, where they are exposed to strong wave action and plenty of sunlight. Their fast growth and ability to regenerate quickly make them well-suited for these dynamic environments. Foliose corals are often found on the reef flat and in the back reef areas, where the water is calmer and light levels are moderate. Table corals are also commonly found in shallow reef environments, where they can maximize their exposure to sunlight.

These varied growth patterns not only shape the physical structure of coral reefs but also create habitats that support immense biodiversity, making coral reefs, even if representing less than 0.1% of the ocean area, some of the most biologically diverse ecosystems on Earth,

with about 25% of the marine species.

Reproduction

The formation and expansion of coral colonies are driven by a combination of sexual and asexual reproductive strategies, each of which plays a vital role in the growth, survival and spread of coral species.

Asexual reproduction is the primary mechanism by which coral colonies expand and repair themselves. The most common form of asexual reproduction is budding, where new polyps are produced from the body of an existing polyp. These new polyps remain attached to the parent polyp, leading to the growth of a larger, stronger, interconnected colony. The shared calcium carbonate skeleton secreted by these polyps provides the structural foundation for the entire colony.

Another important method of asexual reproduction is fragmentation, in which parts of the coral break off, often due to physical disturbances such as storms or strong currents, emit fragments with polyps that can then reattach to a suitable substrate and form new colonies further away. Species like *Acropora* spp. and *Pocillopora* spp. are known for their ability to reproduce through fragmentation, making them proliferic in dynamic environments. Similarly, polyp bail-out, where individual polyps detach from the colony in response to environmental stress allows corals to survive adverse conditions and spread its colonies. Species such as *Fungia* spp. and *Scleractinia* spp. have been observed to reproduce through polyp bail-out.

Sexual reproduction is essential for generating new colonies and ensuring genetic diversity within coral populations. Many coral species, particularly those that are broadcast spawners, engage in mass spawning events. During these events, numerous corals simultaneously release eggs and sperm into the water column, where fertilization occurs externally. The resulting larvae (planulae), are carried by ocean currents, sometimes over great distances, before settling onto a suitable substrate where they can develop into new polyps and establish a new colony. The precise synchronization of these spawning events, while not fully understood, is thought to be influenced by environmental elements such as lunar cycles, water temperature, and daylight length. Species such as *Acropora* spp. and *Porites* spp. are typical broadcast spawners, relying on this strategy to disperse their offspring and colonize new areas of the reef.

Some coral species are brooders, where fertilization occurs internally within the polyp. The larvae are retained inside the polyp until they are more developed and ready to settle soon after being released. This strategy often results in the larvae settling near the parent colony, contributing to the growth of dense coral clusters. *Pocillopora* spp. and *Stylophora* spp. are examples of brooding corals that employ this strategy, leading to the formation of tightly clustered colonies that enhance the local complexity of the reef.

Together, these reproductive strategies ensure the survival, expansion, and genetic diversity of coral populations. They enable coral reefs to adapt to changing environmental conditions, recover from disturbances by colonizing new areas and reinforce existing structures.

Corals, animals and vegetation interactions

Coral reefs are dynamic ecosystems where corals, algae, and reef-dwelling animals are constantly interacting, with each playing a vital role in the health and stability of the reef. The competition between corals and algae is a fundamental aspect of these interactions. Both corals and algae compete for the same resources: sunlight, nutrients, and substrate. However, their growth strategies differ significantly. Corals are slow-growing, reef-building organisms

that provide the structural foundation of the reef, while algae are generally fast-growing and can quickly colonize available space, especially when nutrient levels are elevated due to pollution or when populations of herbivores, such as fish and sea urchins, are reduced due to overfishing or diseases. This algal overgrowth can suffocate coral colonies, blocking sunlight and hindering coral growth and reproduction. A shift in the ecosystem from a coral-dominated system to an algal-dominated significantly reduces the structural complexity of the reef, as algae do not build the rigid frameworks that corals do. The loss of coral cover and the dominance of algae decrease the reef's ability to provide habitat for marine life, leading to a decline in biodiversity and altering the dynamics of the entire ecosystem.

In addition to controlling algae, certain reef animals contribute to coral health through bioerosion, a process where organisms like parrotfish and certain invertebrates break down dead coral structures, which help recycle calcium carbonate within the reef ecosystem. The fine sediment produced by bioerosion can fill in gaps between living coral colonies, stabilizing the reef and providing a more solid foundation for new coral growth.

Fishes also contribute to coral preservation through nutrient cycling. As fishes move through the reef and excrete waste, they release nutrients that can be utilized by corals and other reef organisms. These nutrients can be absorbed by corals to grow, especially in nutrient-poor environments where external nutrient sources are limited. Furthermore, the physical presence of fish within the reef can enhance water circulation, which helps distribute nutrients and oxygen throughout the reef, supporting coral health and resilience.

Corals have evolved various defense mechanisms to protect themselves from algal overgrowth and other threats. Some corals produce chemical compounds that deter algal growth or attract herbivorous fish that feed on algae. Additionally, certain coral species engage in mutualistic relationships with small fish or invertebrates that help defend the coral from algal encroachment and predation.

Environmental changes, particularly those driven by climate change, can disrupt the balance between corals and algae. Warmer ocean temperatures and acidification by increased CO_2 can increase the frequency and severity of coral bleaching events, weakening corals and giving algae a competitive advantage. Additionally, nutrient enrichment from agricultural runoff, sewage, and other sources can lead to algal domination on coral reefs.

Fish and other marine organisms derive significant benefits from living in coral reef habitats, which are essential for their survival, growth, and reproduction. Coral reefs offer a complex three-dimensional structure that provides shelter and protection from predators, particularly for juvenile fish. This protective environment increases the chances of survival for many species.

Additionally, coral reefs serve as critical breeding and nursery grounds. The sheltered environments within the reef are ideal for spawning and raising young, offering protection to eggs and larvae. Coral reefs also facilitate important social interactions, such as territory establishment and mating behaviors, which are vital for reproduction and species survival.

Lastly, some fish species engage in mutualistic relationships with corals, such as clownfish with anemones, gaining protection while providing benefits like nutrient exchange or cleaning services. These interactions underscore the interdependence of fish and corals within the reef ecosystem.

Coastal impact

Coral reefs serve as vital natural barriers for coastlines, acting as a first line of defense against oceanic forces. These complex structures, composed of the reef crest, fore reef, and back reef, significantly reduce incoming wave energy, thereby protecting coastal ecosystems

and communities from storms, waves, and rising sea levels. Coral reefs are especially important for densely populated coastal regions, where they provide crucial protection, as evidenced by the percentage of coastlines safeguarded by reefs globally.

[ADD PROFILE IMAGE FOR FORE-REEF, CREST AND BACK REEF]

A key protective function of coral reefs is their ability to dissipate wave energy. As waves approach the shore, the reef crest acts as a primary barrier, breaking and reducing their intensity. The slope and structural complexity of branching corals contribute to slowing down and dispersing wave energy. This reduction in wave energy directly impacts the shoreline, by reducing coastal erosion and preserving beaches. The stability provided by reefs extends to adjacent coastal ecosystems, such as mangroves and seagrasses, which are also vital for shoreline protection. Together, these ecosystems form a multi-layered defense system where each component enhances the protective functions of the others.

Storm surges have the potential to cause severe coastal flooding. Coral reefs act as natural buffers, reducing the height and energy of these surges before they reach the coast. However, storms also affect the physical structure of the reef itself, with different types of corals responding differently to storm impacts. In regions where coral reefs have remained healthy, communities have benefited from reduced damage during storms, whereas areas with degraded reefs have experienced significant flooding and destruction.

Coral reefs also provide long-term protection against sea-level rise through their ability to grow vertically by depositing calcium carbonate. The sediment stabilization and the protection of mangroves and seagrass areas reduces the erosion on the coasts, the loss of land and flooding. The ability of a reef to respond to sea-level rise is often described in terms of "keep up," "catch up," and "give up." Reefs that "keep up" grow vertically at a rate that matches or exceeds the rate of sea-level rise, ensuring that they continue to provide effective coastal protection. Reefs that "catch up" grow slower than the current rate of sea-level rise but may eventually match it, potentially restoring their protective capabilities over time. On the other hand, reefs that "give up" are unable to grow fast enough to keep up with rising sea levels, leading to their submersion and loss of protective function.

Coral reefs' shape

In Darwin and Jackson, *The Structure and Distribution of Coral Reefs: Being the First Part of the Geology of the Voyage of the 'Beagle,' under the Command of Capt. Fitzroy, R. N., during the Years 1832 to 1836* (1842) , Darwin outlined his atoll reefs formation theory, inspired by his Beagle voyage. He proposed that Earth's crust movements under the oceans created atolls. Darwin described a three-stage process in atoll formation: a fringing reef forms around a sinking volcanic island, evolving into a barrier reef, and finally an atoll reef as subsidence continues.

Fringing reefs

Fringing reefs are the most common and widespread type of coral reef, typically found in close proximity to the shore, forming a narrow band that runs parallel to the coastline. These reefs are directly attached to the land, often with no significant separation between the reef and the shore, or they may be separated by a shallow, narrow lagoon. The proximity of fringing reefs to the coast makes them easily accessible, but it also exposes them to various environmental stressors from both the land and the sea.

The structure of fringing reefs is defined by several key components. The reef flat is the area closest to the shore, characterized by shallow waters where corals grow in a relatively calm environment. The reef flat often extends out to the reef crest, which is the highest point of the reef and is typically exposed at low tide. The reef crest is subjected to the full force of

incoming waves, making it a zone of high energy where only the hardiest coral species can thrive. Beyond the reef crest, the reef slope descends into deeper water, where the diversity and density of coral species often increase. The reef slope provides a habitat for a wide variety of marine organisms, taking advantage of the increased light availability and water movement.

Fringing reefs form in areas where the environmental conditions are favorable for coral growth, including clear, shallow waters with high light availability, warm temperatures, and moderate wave action. These conditions allow corals to establish themselves close to the shore, where they can benefit from the nutrients carried by coastal currents. However, the proximity of fringing reefs to land also makes them more vulnerable to the impacts of sedimentation, freshwater runoff, and pollution from human activities. Sedimentation can smother corals, blocking the light they need for photosynthesis, while excess nutrients from runoff can lead to algal blooms that compete with corals for space and resources.

Within fringing reefs, there is a distinct pattern of zonation, where different coral species and marine organisms are distributed according to the environmental conditions across the reef. On the reef flat, where conditions are relatively calm, corals tend to be more resilient to temperature fluctuations and sedimentation. On the reef crest, the coral species are more adapted to withstand the wave action and exposure during low tides. The reef slope, with its deeper waters and more stable conditions, often supports a greater diversity of coral species, including those that form large, complex structures that provide habitat for a variety of marine life.

Barrier reefs

These reefs typically develop from fringing reefs that have grown over long periods, during which the coastline has either subsided or sea levels have risen. As the land subsides or the sea level rises, the original fringing reef is gradually separated from the shore, forming a lagoon between the reef and the coastline. The depth and size of the lagoon, as well as the height of the reef crest, are influenced by factors such as wave action, currents, and the rate of coral growth. The lagoon often contains patch reefs, seagrass beds, and sandy areas. The zonation within barrier reefs creates distinct ecological niches, with different species of corals, fish, and invertebrates adapted to the specific conditions found in each zone. The fore reef slope, with its high-energy environment, supports species that are resilient to strong waves and currents. In contrast, the more sheltered back reef and lagoon provide habitats for species that prefer calmer waters and stable conditions.

Barrier reefs also play role in the connectivity between marine ecosystems. They act as a bridge between the open ocean and the coastal environments, facilitating the movement of marine species across different habitats. Many species of fish and invertebrates migrate between the reef and the lagoon during different life stages, using the barrier reef as a nursery or feeding ground.

Atolls

The formation of atolls is a process that unfolds over millions of years. As the island gradually subsides, mainly due to tectonic activity, the coral continues to grow upward toward the sunlight. Eventually, the island disappears entirely beneath the ocean's surface, leaving behind a ring-shaped coral reef that encircles the central lagoon. This theory of atoll formation was first proposed by Charles Darwin and remains one of the most widely accepted explanations for the development of atolls.

The lagoon varies greatly in depth, ranging from shallow areas with sandy bottoms to deeper sections where patch reefs and seagrass beds may develop. The sheltered waters of the lagoon provide a calmer environment compared to the outer reef slope, allowing a

different community of marine species to develop. These species often include a mix of corals, fish, and invertebrates that are weakly tolerant of the high-energy conditions. The lagoon may also contain small coral reef islands, known as motus, which form from the accumulation of coral debris and sand.

The outer reef slope, with its exposure to the open ocean, is home to species that are adapted to strong currents and waves, including large predatory fish and robust coral species. In contrast, the sheltered lagoon provides a haven for more delicate species, such as seagrasses, small fish, and invertebrates, which rely on the calmer waters for feeding and reproduction. This diversity of habitats within a relatively small area makes atolls particularly important for marine biodiversity.

Other types of coral reefs and structures

Patch reefs are smaller, isolated coral formations that occur within lagoons, between larger reef systems, or in shallow coastal waters. These reefs are typically found in the sheltered environments of lagoons, where conditions are favorable for coral growth but do not support the development of larger reef structures.

Patch reefs are characterized by their simplicity and isolation compared to their larger counterparts. These small, often circular or irregularly shaped coral assemblages form in areas where the physical and environmental conditions, such as water depth, light availability, and wave action, allow for the localized growth of coral colonies.

The formation of patch reefs is influenced by a combination of biological and physical factors. Coral larvae settle in suitable areas where they can establish colonies and grow over time. They often develop in locations where the water is clear, and the substrate is stable, such as on sandy bottoms or rocky outcrops.

Spurs and grooves are alternating ridges (spurs) and channels (grooves) found on the fore reef slope, which is the part of the reef that faces the open ocean. This distinctive pattern is formed primarily by the action of waves and currents, which erode and shape the coral over time. Spurs, the ridges of coral that extend seaward, help to break up and dissipate the energy of incoming waves. The grooves, or channels between these ridges, serve as pathways for water and sediment to flow back into the ocean after waves break on the reef. This spurred and grooved topography is crucial for the reef's ability to protect the coastline, as it reduces the energy of waves before they reach the more fragile parts of the reef and the shore.

Pinnacles are vertical, column-like coral formations that rise from the seafloor, often found on the fore reef slope or within lagoons. These towering structures add significant three-dimensional complexity to the reef system. Pinnacles can vary greatly in size, from small outcrops to large, towering columns that reach close to the water's surface. The formation of pinnacles is often influenced by localized environmental conditions, such as strong currents or the availability of hard substrate for coral larvae to settle on.

Bommies are isolated outcrops or large coral heads that rise from the seafloor, typically within lagoons or on the reef slope. These features, sometimes referred to as coral knolls, can be quite large and are often found scattered across the lagoon or near the outer edges of the reef. Bommies are important structural components of the reef as they contribute to the overall topographic complexity of the environment.

CHAPTER 2

Automatic generation of coral reef islands

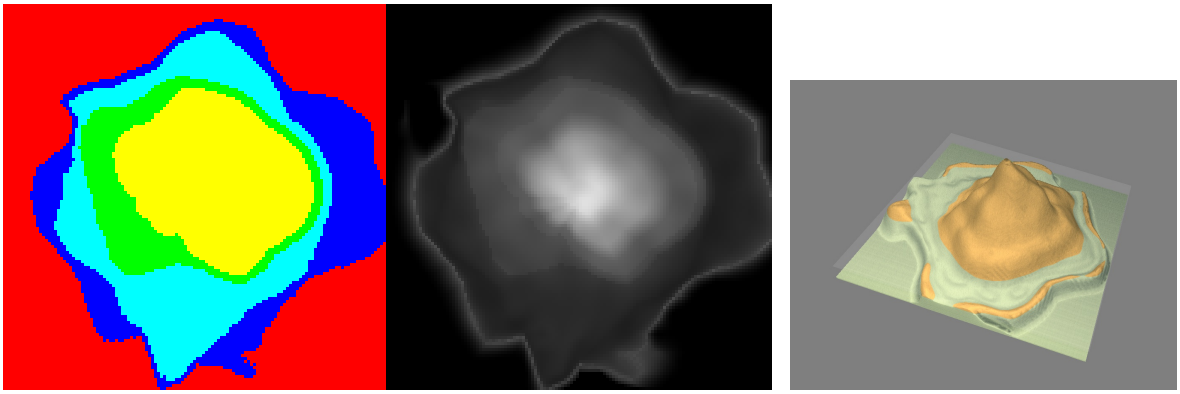


Figure 2.1: Caption.

Abstract

In this chapter, we propose a procedural method for generating single circular volcanic islands using user sketching from two perspectives: a top view, which defines the island's shape (horizontal dimensions), and a profile view, which outlines its elevation and terrain (vertical dimensions). These perspectives, commonly used in geological and remote sensing domains, are complemented by a user-defined wind field, applied as a distortion field to deform the island's shape, mimicking the effects of wind and waves on the long term. We then model the growth of coral on the island and its surronding in a second time [???] to construct the reef following biological observations. Based on these inputs, our method generates a height field of the island. This algorithm is capable of creating many different island models, and we have generated thousands of exemplars to compose the dataset used for training a conditional Generative Adversarial Network (cGAN). By applying data augmentation, the cGAN allows for even greater variety in the generated islands, providing users with more control over the shape and structure of the final output.

Contents

Back to summary

2.1 Introduction

Coral reef islands are formed through a dynamic interplay of volcanic activity, coral growth, and long-term geological processes. These islands begin as volcanic landmasses, created when magma from the Earth's mantle erupts through the ocean floor and builds up layers of volcanic rock, eventually rising above sea level. In tropical waters, these volcanic islands create ideal conditions for coral reefs to develop. Corals, thriving in the shallow, sunlit waters around the island, initially form fringing reefs attached to the island's coastline.

As time passes, the volcanic island undergoes subsidence, a slow sinking process caused by the cooling and contraction of the Earth's crust beneath the island. In response to this subsidence, corals continue to grow upward, maintaining their position within the photic zone, where sunlight supports their survival. This upward growth leads to the formation of barrier reefs, which become separated from the island by a lagoon as the island sinks further.

Eventually, the volcanic island may submerge completely beneath the ocean's surface, leaving only the coral structure visible above water. This process results in the formation of atolls, which are ring-shaped reefs encircling a central lagoon. Over geological time, the physical structure of the island evolves from a prominent volcanic peak to a coral-dominated reef system, shaped by the combined forces of subsidence, coral growth, and erosion.

Charles Darwin's subsidence theory, developed from his observations in Darwin and Jackson, *The Structure and Distribution of Coral Reefs: Being the First Part of the Geology of the*

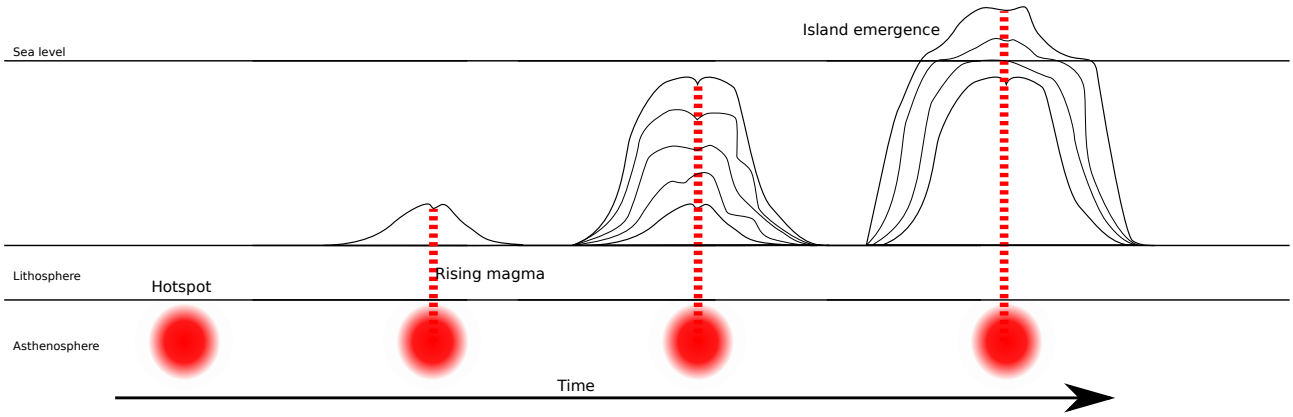


Figure 2.2: Volcanic islands rise above hotspots. The rise of magma from the hotspot forms a seamount. Consecutive eruptions from the volcano grow the seamount until the peak emerge above the sea level. The ground above and close to sea level is affected by hydraulic, aeolian and coastal erosion.

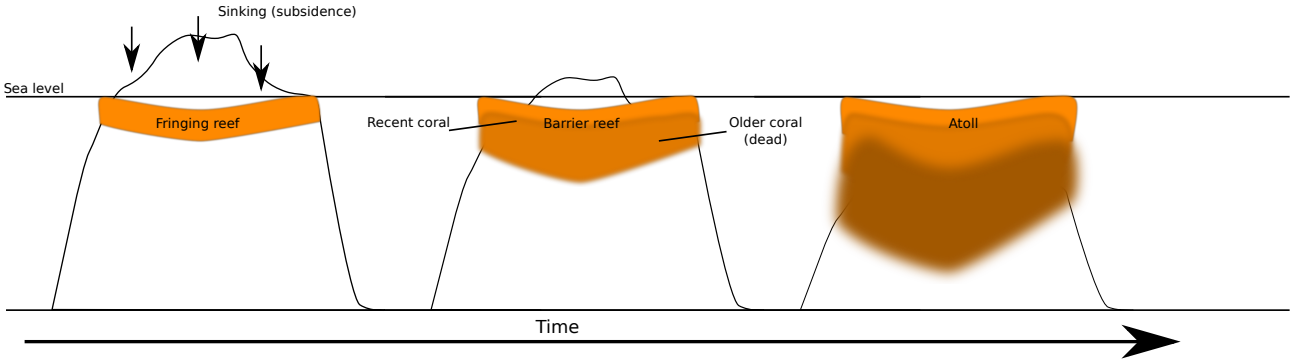


Figure 2.3: Coral colonies grow near sea level, forming a fringing reef. As the hotspot below the island has moved, the island sinks slowly. The coral reef, however, continues its growth to keep living coral colonies in the photic zone. The sinking island increase the size of the lagoon, forming a barrier reef island. When the peak of the island is below the surface of the lagoon, an atoll is formed.

Voyage of the 'Beagle,' under the Command of Capt. Fitzroy, R. N., during the Years 1832 to 1836 (1842) , remains one of the most widely accepted explanations for the formation of coral reef islands. Darwin proposed that coral reefs form around volcanic islands that slowly subside over time due to geological processes. As the volcanic island sinks, coral reefs grow upward, maintaining their position near the surface of the ocean. This theory explains the transition from fringing reefs attached to the island's coast, to barrier reefs separated by a lagoon, and finally to atolls, where the volcanic island has completely submerged, leaving only the coral structure visible above water.

While other theories, such as John Murray's growth on submerged mountains (**MurrayCoralTheory**) [CITATION] and Reginald Daly's sea level change theory (Daly, 1915), have also been proposed to explain coral reef formation, Darwin's subsidence theory remains the most widely supported due to its ability to account for the full evolution of coral islands, from volcanic landmasses to atolls, iven if it has been recently partially disproved by Droxler and Jorry, *The Origin of Modern Atolls : Challenging Darwin's Deeply Ingrained Theory* (2021) [ADD DETAILS]. Its simplicity and focus on the relationship between subsidence and coral growth make it

especially well-suited for computational modeling.

In our approach, we translate the core principles of Darwin's theory into a procedural generation model, simulating the gradual sinking of volcanic islands while coral reefs grow to keep pace with changing sea levels. This allows us to realistically model the transformation of islands from volcanic landmasses to coral-dominated atolls. By capturing this interplay, we can procedurally generate a wide variety of island structures that reflect real-world geological processes.

Simulating the formation of coral reef islands presents significant challenges due to the complex interplay of geological, environmental, and biological factors. One major difficulty lies in capturing the long-term subsidence of volcanic islands, which occurs over millions of years, while simultaneously modeling the upward growth of coral reefs that rely on environmental conditions such as water depth, temperature, and sunlight. This combination of slow geological processes and dynamic biological growth is difficult to replicate in a computational model.

Additionally, the biological aspects of coral growth are inherently tied to environmental factors. Coral reefs grow only within a specific range of water depth and sunlight, and their growth patterns are affected by the health of the reef ecosystem and the availability of resources. Accurately modeling these biological dependencies in a procedural system is challenging, as the factors governing coral reef development are difficult to generalize.

Existing terrain generation methods, such as Perlin noise-based algorithms or uplift-erosion models, are often ill-suited for these processes. While they can generate natural-looking landscapes, they do not account for the unique geological and biological interactions that govern coral island formation. Capturing these dynamics requires a balance between realism and procedural flexibility, allowing for both accurate simulation of natural processes and intuitive user control.

To address these challenges, we present a procedural generation tool that simulates the formation and evolution of coral reef islands by integrating both geological and biological processes. Our approach allows users to control the island's shape and structure through intuitive sketching interfaces, while the tool models subsidence, coral growth, and environmental deformation.

The generation process begins with two user-defined inputs: a top view to outline the island's overall shape and a profile view to define its elevation and terrain contours. These inputs serve as the foundation for creating a height field of the island. Additionally, a wind deformation field allows the user to simulate the effects of wind and waves, reshaping the island's structure in a way that mimics natural erosion processes.

To enhance flexibility and allow for more complex, non-circular island structures, we incorporate a conditional Generative Adversarial Network (cGAN) into the generation process. The cGAN is trained on a dataset of islands generated by the initial algorithm, augmented to introduce a wider variety of shapes and features. This machine learning component allows for the automatic generation of diverse island models, removing some of the constraints of the initial procedural algorithm such as strictly radial symmetry while maintaining user control over key aspects of the design.

By combining user input with the adaptive power of the cGAN, our tool generates plausible coral reef islands that evolve from volcanic landmasses to coral-dominated atolls, capturing both geological and environmental processes.

2.2 Overview

Our system for generating coral reef islands combines user-driven sketching, procedural techniques, and deep learning to create realistic and varied island terrains. The process begins with the user sketching key features of the island, such as its overall shape and profile. This sketching process allows the user to define the layout of the island in an intuitive way, providing control over the placement of important elements like the island borders, beaches, lagoons, and coral reefs.

[ADD DETAILS ON THE SIMULATION PART]

Once the user's sketch is complete, the system converts these high-level features into a labeled map, where each pixel is assigned to a specific region of the island. This map serves as the input to a conditional Generative Adversarial Network (cGAN), which is responsible for generating the fine details of the terrain. The cGAN has been trained on a dataset of island examples generated by the initial procedural algorithm, allowing it to produce realistic terrains that reflect the user's design while introducing natural variation and complexity.

By conditioning the cGAN on the user-defined map, the system maintains the overall structure and regions specified in the sketch, while generating realistic transitions between different areas (such as beaches, lagoons, and the island's interior). The use of deep learning, in combination with procedural techniques, allows the system to create coral islands that are both geologically plausible and tailored to the user's input.

This method provides a simple and flexible approach to island generation, enabling the creation of diverse island structures that reflect real-world geological processes, such as volcanic subsidence and coral reef growth.

2.3 State of the art

Procedural generation of terrain has been a well-researched area in computer graphics and simulations, where the goal is to create large, realistic landscapes with minimal manual input. Various methods have been developed over the years to generate terrains automatically, from noise-based approaches to physically-based erosion simulations, sketch-driven methods, and more recently, deep learning techniques.

However, each of these techniques has its strengths and limitations, particularly when it comes to modeling coral reef islands. Coral islands present unique challenges due to the combination of long-term geological processes (such as subsidence and coral reef growth) and environmental interactions (like erosion caused by wind and waves). In this section, we review the key techniques that have been applied to terrain generation, highlight their limitations for coral island formation, and position our work as an approach that addresses these challenges.

2.3.1 Noise functions

Noise-based procedural generation remains one of the most widely used techniques for creating natural-looking terrains. Introduced by Ken Perlin, the Perlin noise and Simplex noise are foundational algorithms that generates pseudo-random yet continuous variations across a grid, producing terrain features that resemble organic landscapes (Perlin, 1985; 2001).

Beyond basic noise functions, more advanced techniques, such as fractal Brownian motion (fBm), are commonly used to add additional detail to terrains. FBm combines multiple layers,

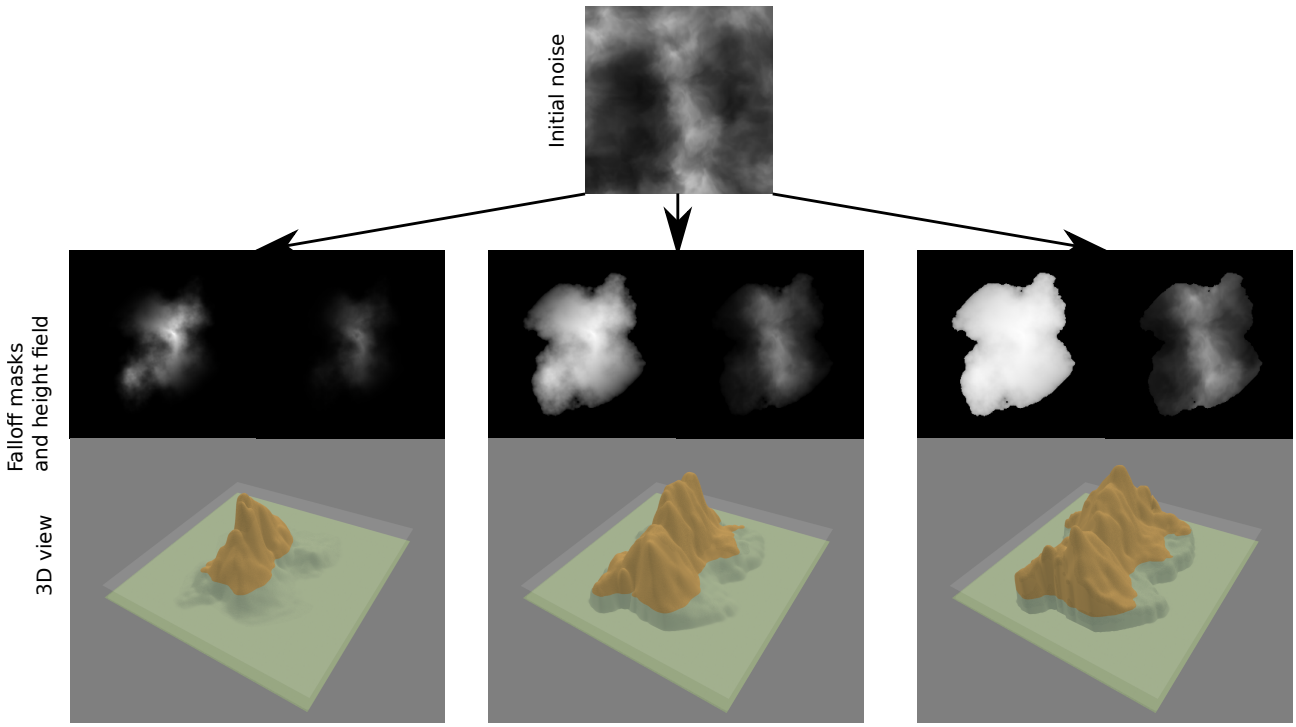


Figure 2.4: Three different results of an island generated from noise functions. In each case, the initial height field is the same, computed through flow noise. The falloff masks are also generated with a combination of fBm noise mitigated by the euclidean distance from the image center, warp noise and gamma correction. The only parameter modified for each example is the gamma correction. The results are very different, and hardly controllable. It is also difficult to represent lagoons and reefs using this method.

or "octaves," of noise at different frequencies, creating terrains with finer details and more realistic features. Noise functions are often paired with falloff maps to generate island-like terrains, where the height of the terrain gradually decreases toward the edges, mimicking the appearance of coastlines and providing a simple way to create basic island shapes.

Despite their widespread use, noise-based methods have significant limitations when applied to the simulation of coral reef islands. While these techniques excel at producing large, varied landscapes quickly, they lack the geological accuracy needed to model real-world processes like volcanic subsidence and coral growth. These approaches generate random patterns, but are disconnected from the actual physical processes that govern coral island formation.

Our approach goes beyond the randomness of noise-based generation by incorporating real-world geological and biological processes into terrain formation. Specifically, we model the gradual subsidence of volcanic islands and the upward growth of coral reefs, which are essential for representing the long-term evolution of coral reef islands. By integrating these processes into the generation algorithm, we create terrains that are not only more realistic but also allow for user control. This blend of user input and geological grounding provides a more accurate and flexible approach to modeling coral reef islands, addressing the limitations inherent in traditional noise-based techniques.

2.3.2 Erosion simulation

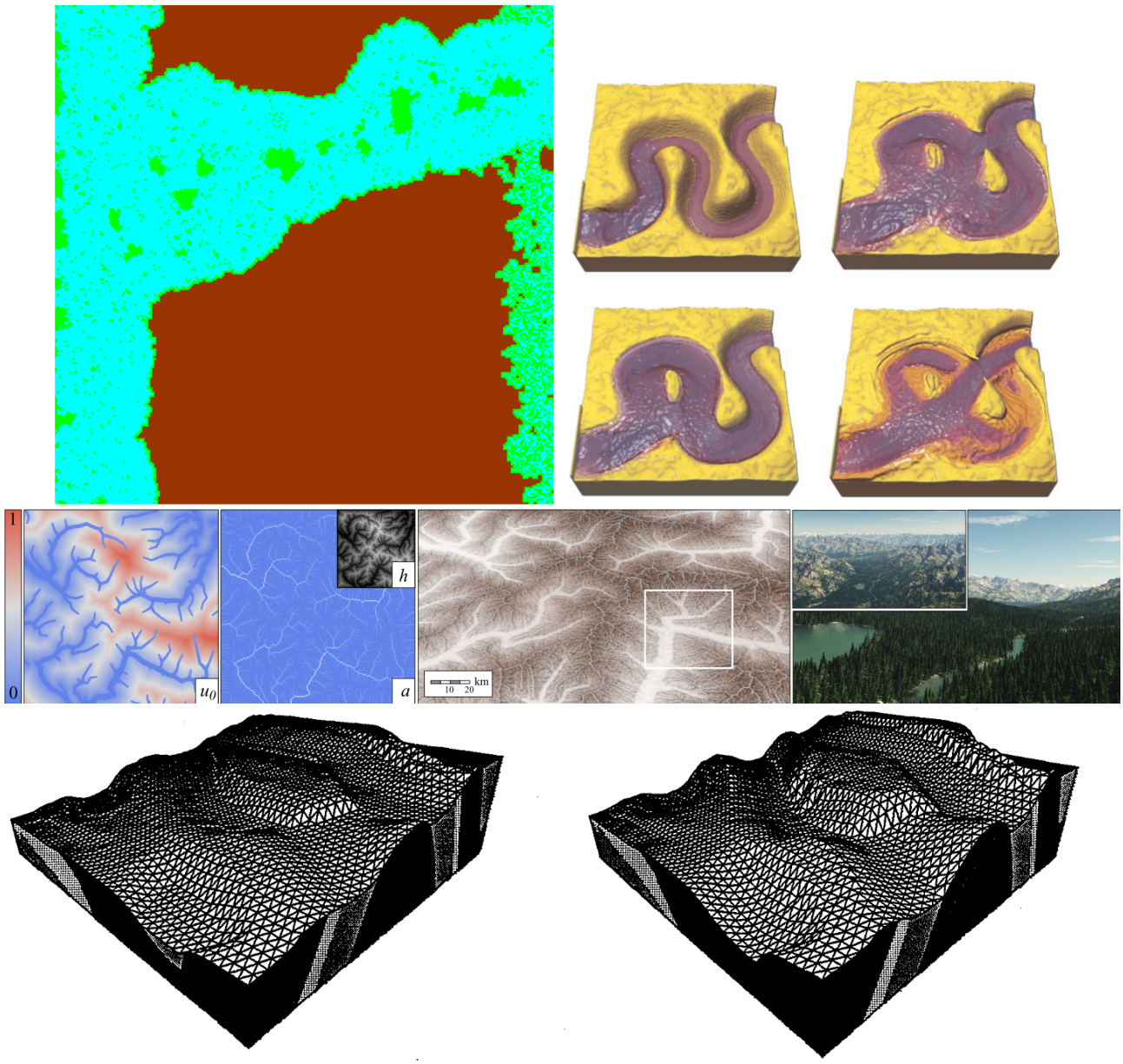


Figure 2.5: Four examples from (Hawick, 2014), (Beneš, Těšínský, et al., 2006), (Schott et al., 2023) and (Roudier et al., 1993).

While noise functions generate random, natural-looking terrain, they often fail to capture the physical processes that shape real landscapes over time. To address this limitation, erosion simulations model how natural forces such as water flow and gravity wear down terrain features, creating valleys, river networks, and other detailed landforms. These simulations are particularly effective at representing mountainous landscapes, where erosion gradually sculpts the terrain over long periods of time, typically on the scale of hundreds to millions of years.

Erosion simulations often use models like the Stream Power Law to describe how water erodes higher elevations and deposits sediment in lower areas. The rate of erosion depends on factors such as slope steepness and water flow, with steeper slopes and greater water volumes

leading to faster erosion. Over time, these processes create detailed and realistic terrain features like river valleys and erosion channels. Erosion models are iterative, simulating the gradual shaping of landscapes by continuously balancing forces like erosion and uplift, as seen in the work of (Cordonnier, Braun, et al., 2016; Cordonnier, Cani, et al., 2017). Their model simulates how tectonic forces uplift terrain, which is then eroded away over thousands of years. Similarly, (Schott et al., 2023) and (Tzathas et al., 2024) refined the Stream Power Law to generate realistic slopes and ridges through these long-term geological processes.

Despite their effectiveness in modeling large-scale terrain evolution, erosion simulations are limited when it comes to representing environments that are shaped by dynamic biological processes. Coral reefs, for instance, grow and adapt to changing environmental conditions on a much shorter timescale than geological erosion. While erosion models simulate terrain changes over millennia, coral growth responds more dynamically to factors like water depth, light availability, and temperature on scales ranging from years to decades. These biological factors are difficult to incorporate into traditional erosion models, which are designed to simulate the slow, steady forces of water and gravity. Recent works however take into account biologic factors in the erosion simulation (Cordonnier, Galin, et al., 2017) [CITATION DEADWOOD DE GUERIN 2024] [ADD DETAILS, WHAT IT CAN OR CAN'T DO, PROBLEM OF TIME].

Our approach builds on the concept of terrain evolution by integrating the geological and biological dynamics that shape coral reef islands. Instead of focusing solely on erosion, which is primarily a surface-level process, we model the subsidence of volcanic islands over geological time and the simultaneous growth of coral reefs that adapt to the changing sea levels. Coral reefs must grow upward to remain within the photic zone, a process that unfolds on shorter timescales than those captured by standard erosion simulations. Furthermore, we introduce wind deformation to simulate the effects of environmental forces like wind and waves, which reshape the island's structure. This combination of long-term geological processes and short-term biological responses provides a more comprehensive simulation of coral reef island formation, capturing both the slow subsidence of volcanic landmasses and the rapid adaptability of coral ecosystems.

2.3.3 Sketching

In procedural terrain generation, sketch-based approaches allow users to directly interact with and shape landscapes through intuitive sketching interfaces. These methods enable users to define key terrain features such as mountains, valleys, and coastlines by drawing them in a two-dimensional canvas, which are then translated into 2.5D terrains. Sketching provides a high level of artistic control, making it especially useful in creative applications such as video games and simulations, where user-defined terrain features are prioritized.

Sketch-based systems offer great flexibility, allowing users to directly define specific elements of the landscape according to their needs or preferences. For instance, Gain et al., *Terrain sketching* (2009) introduced a multi-perspective sketching system that allows users to generate detailed landscapes by sketching from different angles. This method provides more control over the terrain's shape than traditional noise-based generation methods. Similarly, Tasse, Emilien, Cani, Hahmann, and Bernhardt, *First Person Sketch-based Terrain Editing* (2014) explored sketching from the player's viewpoint, allowing users to dynamically define height information from within the virtual environment, creating an interactive terrain design experience.

Sketching has also been extended to geological modeling. Patel et al., *Modeling Terrains and Subsurface Geology* (2021) proposed a system where users can interactively model

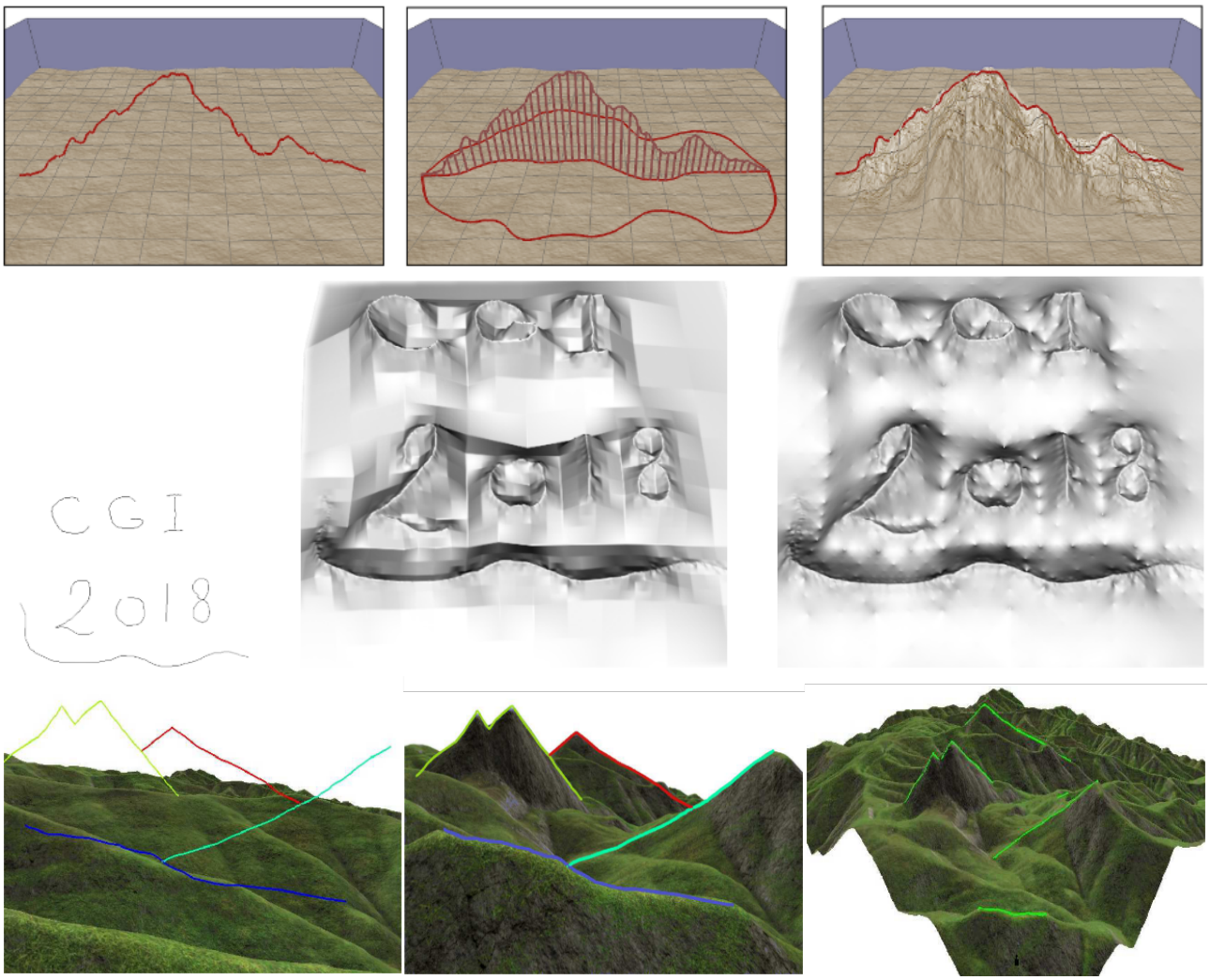


Figure 2.6: From (Gain et al., 2009), (Talgorn and Belhadj, 2018), (Tasse, Emilien, Cani, Hahmann, and Dodgson, 2014)

underground layers by sketching subsurface structures. This provides a powerful tool for geologists and designers who need to model complex, multi-layered terrains, offering more control over both the surface and the subsurface.

In our work, we leverage the flexibility of sketching to define the key features of coral reef islands, such as the island shape, lagoons, and coral reefs, in a vectorized format. This is particularly important for modeling underwater and island landscapes, where these features are not simply surface elements but part of an evolving geological structure. By using sketches, we retain the semantic information of where the island and coral regions are located, which allows us to later apply our simplified model of Darwin's subsidence theory.

Rather than focusing on detailed long-term or short-term evolution, our method adapts sketching to the unique requirements of island and underwater landscapes. Once the user defines the terrain layout through sketching, we apply a simplified subsidence model, where the volcanic island gradually sinks and coral reefs grow in response, remaining close to the water surface. This process provides a framework for simulating the evolution of the landscape in a geologically plausible manner.

The key advantage of sketching in our system is that it combines the artistic control provided by traditional sketching methods with the ability to handle the geological processes

specific to coral reef islands. By preserving the vectorized information from the sketch, we can accurately place and evolve features like lagoons and coral reefs, ensuring that the resulting terrain reflects both the user's design intent and the geological dynamics at play.

2.3.4 Deep learning

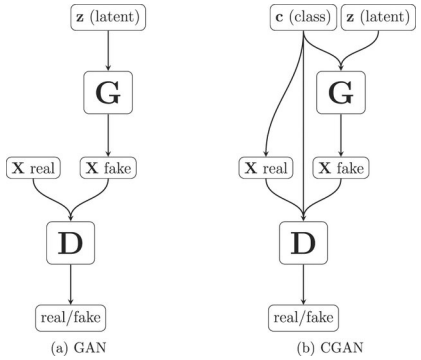


Figure 2.7: The general structure of GAN and cGAN networks are similar: a generator network G is trained to take some noise z as input to try to create a "realistic" output X_{fake} and a discriminator network D is trained parallelly to distinguish generated data from real data. cGAN networks introduce an information of class c in the input, which is used by the generator and discriminator in their inference process. In the end, only the generator is used to create new data.

In recent years, deep learning has become a innovative [MEH... "GAME CHANGER" OR SOMETHING LIKE THIS] tool in computer graphics, image processing, and procedural content generation. Deep learning refers to a class of machine learning techniques that use artificial neural networks with multiple layers to model complex patterns in data. By training these networks on large datasets, deep learning models can learn to generate new, realistic data by identifying the underlying relationships between inputs and outputs.

In terrain generation, deep learning is particularly well-suited for tasks that require the creation of complex, natural-looking landscapes. Traditional procedural generation techniques like noise functions and erosion simulations can be powerful, but they often struggle to produce the complexity and realism of natural terrains due to their reliance on predefined rules and algorithms. In contrast, deep learning models, especially Generative Adversarial Networks (GANs), can learn from real-world data, enabling them to generate realistic terrains that mimic the diversity and complexity found in nature.

GANs are a particularly effective deep learning architecture for terrain generation. A GAN consists of two neural networks: a generator, which creates new data samples, and a discriminator, which evaluates whether the generated samples are real or fake compared to a set of training data. Through this adversarial process, the generator gradually improves its ability to produce realistic examples, while the discriminator becomes better at distinguishing real from fake data. This feedback loop allows GANs to generate highly realistic and detailed terrain features by learning from large datasets of existing terrains.

In terrain generation, GANs have been applied to generate diverse landscapes by learning the patterns and features found in real-world terrains. For example, Guérin, Digne, Galin, Peytavie, et al., *Interactive example-based terrain authoring with conditional generative adversarial networks* (2017) demonstrated how GANs can be used to transform 2D sketches into 3D terrains by training the network on a dataset of terrain features, allowing the system to generate new landscapes based on simple user input. GANs are particularly well-suited for

this task because they can capture the subtle details of terrain that would be difficult to model explicitly, such as the natural flow of elevation or the transitions between different terrain types.

A variant of GANs, known as Conditional Generative Adversarial Networks (cGANs), takes the power of GANs one step further by allowing the generated data to be conditioned on additional inputs. In a cGAN, both the generator and the discriminator receive additional information such as a class label, a sketch, or an image, that guides the generation process. This makes cGANs particularly useful for terrain generation tasks where users want to control specific aspects of the output while still relying on the model to generate realistic details.

In the case of terrain generation, a cGAN allows the user to specify high-level constraints (such as the regions where different terrain types should be) while the generator fills in the finer details based on what it has learned from the training data. This makes cGANs particularly effective in applications where a mix of user input and data-driven generation is required, as the user can control the broad layout of the terrain while the model ensures that the resulting terrain looks natural and plausible.

In our approach, we use a cGAN to generate coral reef islands, providing a balance between user control and realistic terrain generation. After the initial algorithm outlines the regions of the island such as the island itself, beaches, and lagoons, these regions are transformed into a single image, where each pixel represents a different region ID. This image is used as the input for the cGAN, which conditions the generation process on this initial layout.

The cGAN is trained on a dataset of island examples, which are created by the initial procedural generation algorithm and further augmented to introduce a wide variety of shapes and features. This training allows the cGAN to generate coral reef islands that follow the high-level layout provided by the user while introducing natural variations and details that reflect real-world island characteristics. By conditioning the cGAN on the user-defined region map, we ensure that the generated islands maintain structural coherence while still exhibiting realistic features like smooth transitions between regions, varied terrain, and non-circular shapes.

One of the key advantages of using a cGAN in our system is its ability to overcome procedural constraints. Traditional procedural generation methods often impose limitations like radial symmetry or require islands to follow overly simplified geometries. With the cGAN, these constraints can be lifted. The model can generate irregular, non-circular islands that adhere to the user's input but introduce a level of complexity and realism that would be difficult to achieve with purely procedural methods.

Additionally, the cGAN ensures that the generated island terrain aligns with the geological processes modeled in the system, such as subsidence and coral reef growth. By training the cGAN on thousands of examples that reflect these processes, we ensure that the final generated island models are both flexible and geologically plausible. The combination of user-driven design and data-driven generation through the cGAN allows for the creation of coral reef islands that are not only tailored to the user's input but also realistic in their form and structure.

2.4 Example generation

Generating synthetic coral reef island terrains involves a combination of user input, procedural techniques, and simplified geological simulations. The process begins with the creation of a base terrain using user-defined sketches, followed by a simulation of coral

reef growth and subsidence. These two steps, although interconnected, can also be used independently, providing flexibility in how the system is applied. In this section, we describe the example generation process in detail.

2.4.1 Assumptions

In generating synthetic coral reef islands, we adopt a set of simplifying assumptions that are grounded in geological and biological observations. These assumptions streamline the process while producing plausible terrains. Below, we outline the key characteristics of our model:

Radial symmetry: Volcanic islands often begin as circular landmasses due to the radial uplift of magma from beneath the Earth's mantle. Erosion processes further smooth out sharp edges over time, reinforcing this initial symmetry. In our approach, we assume islands start with an approximately circular shape, providing a straightforward basis for terrain generation.

Radial arrangement of features: The key island features like the island body, beaches, lagoons, and coral reefs, are arranged in concentric regions around the island center. This radial arrangement aligns with natural geological progression from the island to the surrounding reef.

Wind and wave deformation: Over time, wind and wave forces shape islands by introducing concave regions, carving out lagoons, and deforming coastlines [ADD CITATIONS]. To simulate these effects, we apply a wind deformation field that allows users to introduce natural variations that break the radial symmetry, resulting in more realistic island shapes.

Independence of islands: In this model, each island is treated as an independent entity, meaning its terrain does not influence neighboring islands. This allows multiple islands to be generated and placed freely without concern for terrain overlap or interaction.

Uniform profile shape: We assume that the vertical profile of the island remains relatively uniform, meaning that the terrain's cross-section from the center outward is similar in every direction. While not accurate, this allows for consistent scaling of the profile in all directions, ensuring that the island's shape remains coherent and plausible.

Coral growth at constant depth: Coral reefs require sunlight to thrive, which limits their growth to depths where light penetrates the water. In our model, coral growth is constrained to this depth range, with corals maintaining their proximity to the water surface as the volcanic island subsides.

Subsidence and coral growth: As the volcanic island subsides, coral reefs grow to keep pace with the sinking landmass, maintaining their height near the water surface, we call the "keep-up" strategy. In our model, the island subsides as a whole, while coral features are preserved by growing vertically. This separation allows the coral regions to remain unaffected by the subsidence of the island itself.

These assumptions, while simplified, provide an effective foundation for generating coral reef islands. They balance computational efficiency with geological plausibility, ensuring that the generated terrains are both flexible and realistic. Furthermore, the system allows users to break the radial symmetry and introduce more natural variations through wind deformation, adding another layer of flexibility to the model.

2.4.2 User input

The generation of coral reef islands in this system begins with two intuitive sketch-based inputs from the user: a top-view sketch and a profile-view sketch, which define the islands horizontal layout and vertical elevation profile. In addition to these sketches, the user can further refine the terrain by applying wind deformation strokes, which simulate the effects of wind and waves on the islands shape. This combination of sketches and wind inputs gives users precise control over both the islands structure and its natural variations, such as irregular coastlines or concave features. We will present the usefulness of these sketches in this section, and describe the technical details in the next section.

Top-view sketch

The top-view sketch defines the islands outline as seen from above. Using a simple drawing interface, the user can delineate the boundaries between key regions of the island, including the island itself, the beaches, the lagoon, and the surrounding abyss. The system assumes that these regions are arranged concentrically around the center of the island, with each boundary defined by a radial distance from the center.

Each region's boundary is represented in polar coordinates, with $r_{\mathbf{p}}$ indicating the radial distance from the islands center and $\theta_{\mathbf{p}}$ representing the angular position. This polar representation allows the system to map the users sketch onto a circular framework, ensuring smooth transitions between regions and maintaining a coherent layout for the island.

In this sketch, the user defines the overall horizontal layout of the island, including the size and shape of each feature. Variations in the outline are introduced by allowing the radial distances to vary with angle, ensuring that the island is not strictly symmetrical and introducing more natural, irregular shapes.

Profile-view sketch

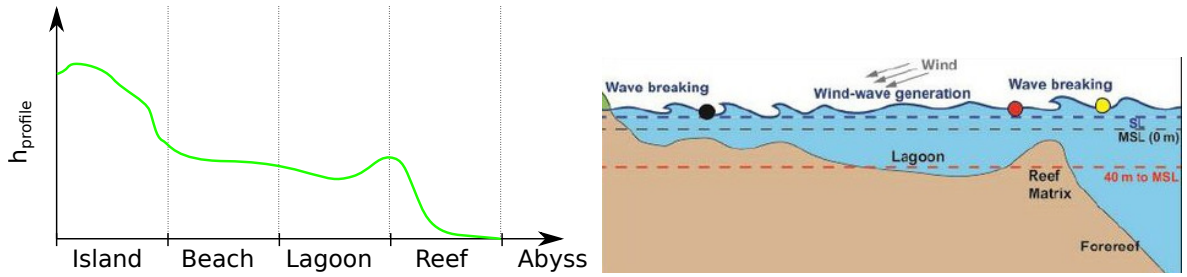


Figure 2.8: (Left) A profile function h_{profile} is defined as a 1D function and represents the surface from the center of the island to the abysses. (Right) The cross-section representation of an island is often represented as a 1D function defined using terrain features as landmarks.

The profile-view sketch defines the vertical elevation profile of the island along any radial direction, offering control over the islands height. In this view, the user specifies the elevation of different regions of the island, such as the island peak, beach, lagoon, abyss, and everything in-between, by drawing the corresponding profile curve.

The regions outlines correspond to key terrain transitions: the highest point of the island (center), the island border, the beach, the lagoon, and the deep-sea abyss. The system uses these milestones to interpolate a continuous 1D height function $h_{\text{profile}}(\tilde{x})$, where $\tilde{x}$ represents

a non-uniform region distance from the islands center, and $h = h_{\text{profile}}(\tilde{x})$ gives the height at each point. This continuous profile ensures smooth elevation transitions across the island.

By combining the top-view and profile-view sketches, the system can generate a full 3D terrain model that accurately reflects the users design by revolution modeling.

Wind velocity field

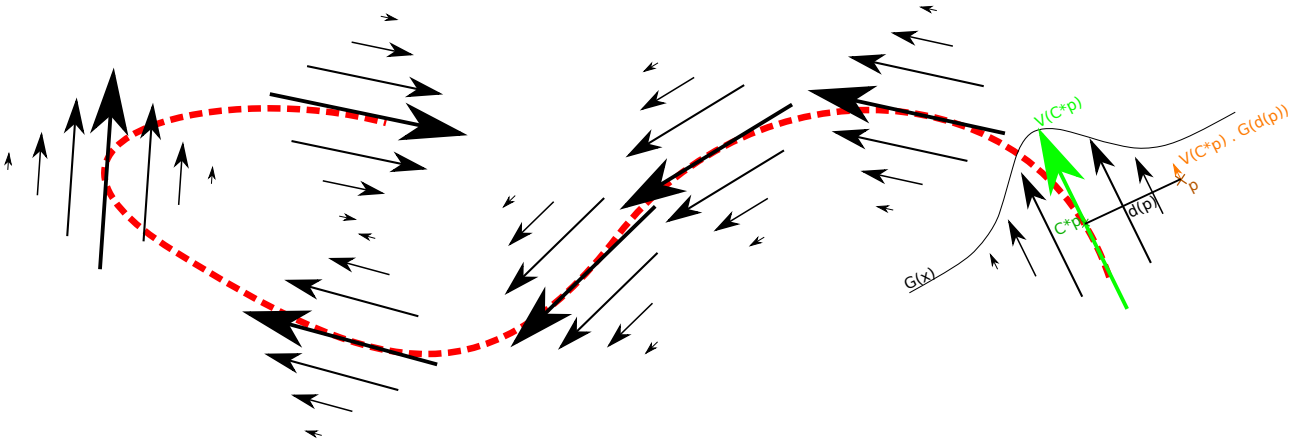


Figure 2.9: From the parametric curve defined by a user (red), we define the velocity field by considering the velocity (first derivative) of the curve at the closest point $\mathbf{p}_C^*$, modulated by a gaussian distance function $G(x)$.

In addition to the sketches, the user can influence the shape of the island by defining a wind velocity field. This field simulates the effects of wind and wave erosion on the island's surface, introducing natural deformations such as coastline indentations, concave features, or variable lagoon shapes.

The wind field is represented as a series of wind strokes drawn by the user on a 2D canvas. Each stroke represents a parametric curve, where the direction and strength of the wind are encoded as a vector field. The user controls the wind's direction by drawing these curves, and the system interprets the strokes to create a velocity field that defines how the terrain should be deformed.

As the user draws a wind stroke, the system generates a set of control points along the curve, with the option to adjust the stroke's width. The width of each stroke determines the area of influence around the curve, where wider strokes result in broader deformations of the terrain. The deformation strength decreases with distance from the wind curve using a Gaussian falloff function using the stroke width as standard deviation, ensuring that the terrain transitions smoothly from deformed regions to non-deformed areas. Once the wind strokes are applied, the system processes the wind velocity field by displacing the terrain points accordingly. The height field, originally generated from the user's sketches, is modified by the wind field to create non-radial features, breaking the initial radial symmetry and producing a more organic island shape.

User interaction

As users draw the top-view and profile-view sketches, the system provides real-time feedback on the resulting terrain. The top-view sketch influences the horizontal layout of the island, while the profile-view sketch defines its vertical structure. These sketches can be

adjusted independently, allowing the user to fine-tune both the outline and elevation of the island.

After sketching the basic shape, users can apply wind deformation strokes to modify the island's features further. These strokes represent wind and wave influences, distorting the island's shape to introduce more natural, non-radial features such as indentations along the coastline, variable lagoon shapes, or concave formations. The system automatically applies these deformations, providing real-time feedback as the user interacts with the terrain.

This interactive process, combining sketches and wind deformation, allows users to quickly iterate on their designs, refining the terrain to meet specific aesthetic or functional goals.

2.4.3 Generation process

The generation of coral reef island terrains involves a structured process that takes the user's sketches and produces a complete 3D terrain model. This process begins with the creation of the initial height field based on the user's input, followed by the application of wind deformation to introduce natural variations, and concludes with the integration of coral reef features through subsidence and coral growth modeling.

Initial height field generation

The generation of the coral reef island terrain begins by transforming the user-defined top-view and profile-view sketches into a coherent 3D height field. This process combines the radial layout of the top-view sketch with the elevation information provided by the profile-view sketch, creating a terrain that accurately represents the desired features, such as the island, beaches, lagoons, and abyss.

For any point $\mathbf{p}$ on the terrain, the system first computes the polar coordinates $(r_\mathbf{p}, \theta_\mathbf{p})$, where $r_\mathbf{p}$ is the radial distance from the island's center, and $\theta_\mathbf{p}$ is the angular component. The radial distance $r_\mathbf{p}$ is used to determine which region the point belongs to (island, beach, lagoon, reef, or abyss). The user-defined outlines in the profile sketch specify the radial limits between these regions.

Each point's height is determined by the profile function $h_{\text{profile}}(\tilde{x})$, where $\tilde{x}$ represents a "piecewise parametric distance" from the island's center. The piecewise parametric distance works by dividing the radial distance from the center into segments, defined by these region boundaries. Each segment corresponds to a distinct region of the terrain, and within each segment, the distance $\tilde{x}$ is interpolated between the region boundaries. For a point $\mathbf{p}$ lying between two boundaries R_i and R_{i+1}, the distance $\tilde{x}_\mathbf{p}$ is calculated as:

$$\tilde{x}_\mathbf{p} = i + \frac{r_\mathbf{p} - R_i}{R_{i+1} - R_i} \tag{2.1}$$

where i is the index of the nearest lower region boundary. This method allows for smooth transitions between regions, even when the spacing between boundaries varies.

For any point $\mathbf{p}$, the height is finally computed as:

$$h(\mathbf{p}) = h_{\text{profile}}(\tilde{x}_\mathbf{p}) \tag{2.2}$$

This approach ensures that the height field accurately follows the elevation profile specified by the user while maintaining smooth transitions between different regions of the island.

The result is a height field that captures both the radial structure of the island (from the top-view sketch) and the vertical elevation profile (from the profile-view sketch), producing a realistic representation of islands with smooth transitions between the key terrain features.

2.4.4 Wind deformation

After generating the initial height field based on the top-view and profile-view sketches, the next step in the process introduces wind deformation. This step simulates the long-term effects of wind and wave erosion, breaking the radial symmetry of the terrain and adding natural variations such as concave coastlines and irregular island shapes.

The wind deformation can be controlled through a user-defined vector field, which represents the direction and strength of wind flows across the terrain. Users interact with the system by drawing strokes on a 2D canvas, which are then interpreted as parametric curves C representing wind patterns. Each stroke defines a wind flow in the curve's direction C' and an effect width σ, and these wind flows are used to displace the terrain, simulating the gradual reshaping of the island due to wind and wave erosion.

The strokes are represented as Catmull-Rom splines, a type of parametric curve that allows for smooth, continuous wind paths. For any point $\mathbf{p}$ on the terrain, the deformation vector $\Phi(\mathbf{p})$ is calculated based on the proximity of $\mathbf{p}$ to the nearest wind strokes. The strength of the displacement is controlled by a Gaussian scaling function, which ensures that points closer to the wind strokes experience stronger displacement, while points farther away are less affected.

The displacement function $\Phi(\mathbf{p})$ is computed as a sum of the influences from all nearby wind strokes. For each stroke, the deformation vector is scaled by a Gaussian function that decreases with the distance from $\mathbf{p}_C^*$ the closest point on the parametric curve C, as follows:

$$\Phi(\mathbf{p}) = \sum_{C \in \text{curves}} C'(\mathbf{q}) \cdot e^{-\frac{\left\|\mathbf{p}-\mathbf{p}_C^*\right\|^2}{2\sigma^2}} \tag{2.3}$$

Once the deformation vector $\Phi(\mathbf{p})$ is computed, the terrain height at point $\mathbf{p}$ is adjusted by displacing $\mathbf{p}$ to a new point $\Phi(\mathbf{p})$. We can then compute the final height $h(\Phi \circ \mathbf{p}) = h_{\text{profile}}(t_{\mathbf{p}})$, or, as the implicit modeling community would write it,

$$\tilde{h}(\mathbf{p}) = \Phi^{-1} \circ h \tag{2.4}$$

This process introduces variations in the terrain, distorting the coastline, creating concave regions, and breaking the original radial symmetry defined by the top-view and profile-view sketches.

Resistance to deformation

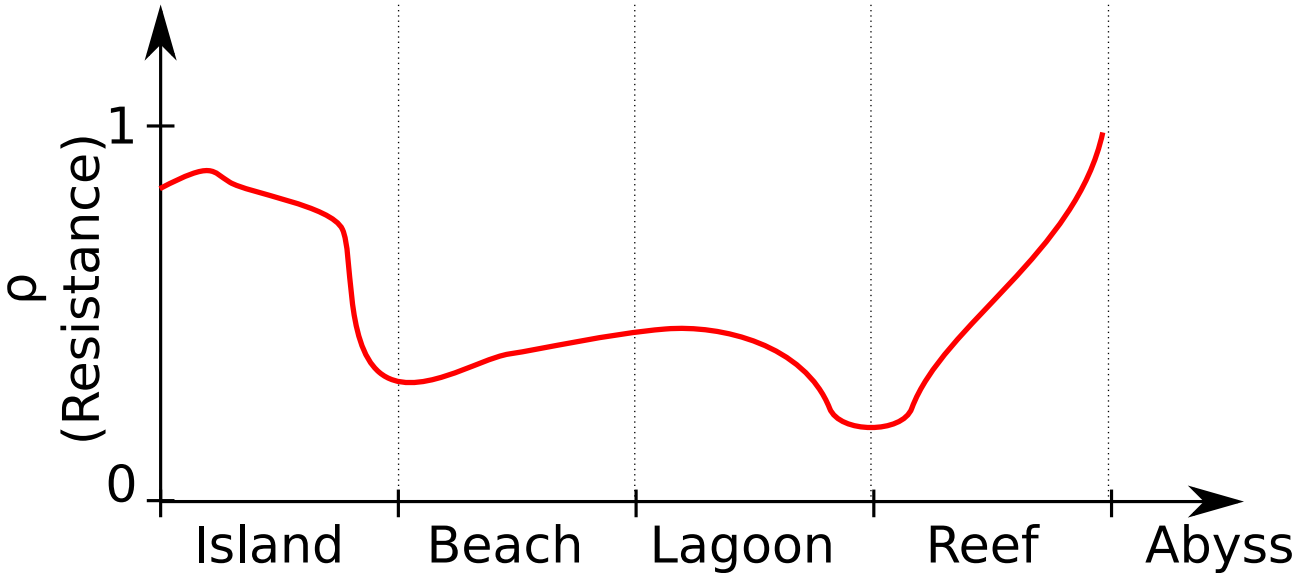


Figure 2.10: The resistance function of the island is defined in the same way than the h_{profile} function. The resistance to erosion and deformation arise from multiple factors such as depth, materials, wind shadowing, biotic and abiotic factors, ... Modeling all these factors is complex. As such, using a user-defined approximation through a resistance function ρ allows for more control.

To ensure that certain regions of the terrain, such as deep-water areas, remain relatively unaffected by the wind, a resistance function $\rho(\tilde{x})$ is applied. The resistance function modulates the effect of the wind deformation based on the previously computed piecewise parametric distance $\tilde{x}$.

The resistance function $\rho(\tilde{x})$ is defined similarly to the profile function, and it controls the magnitude of the displacement at each point. For example, regions near the coastline (such as the beach and lagoon) might have lower resistance, allowing for more significant deformation, while regions farther away (such as the abyss) have higher resistance, limiting the wind and coastal erosion impact.

The deformation vector is scaled by the resistance function at each point $\mathbf{p}$, such that the final deformation vector becomes:

$$\tilde{\Phi}(\mathbf{p}) = \big(1 - \rho(\tilde{x}_{\mathbf{p}})\big) \cdot \Phi(\mathbf{p}) \tag{2.5}$$

Where $\rho(\tilde{x}_{\mathbf{p}})$ is the resistance value corresponding to the point's distance $\tilde{x}_{\mathbf{p}}$. This ensures that the wind deformation has the greatest impact on areas like the coastline and beach, where erosion naturally plays a larger role, while deeper regions like the abyss or stronger regions like mountains remain stable and relatively unchanged.

Deformation and height field update

The wind deformation process results in a modified height field where the terrain has been warped according to the user-defined wind strokes. This deformation introduces non-radial

features, such as concave coastlines or irregularities along the beach and lagoon, making the island appear more natural and varied.

Both the height field and the labeled map (which tracks the terrain regions) are updated to reflect the wind deformation. This ensures that the semantic information of the terrain remains consistent even after the terrain has been warped. The labeled map is deformed in the same way as the height field, preserving the logical structure of the island for further post-processing, such as texturing.

For instance, consider a simple circular island generated from the initial height field. By applying wind strokes along one side of the island, the deformation process can create concave regions along the coastline, making the shape more irregular and mimicking the effects of real-world wind and wave erosion. The resistance function ensures that while the beach and lagoon areas are deformed, the abyss remains largely unaffected as they are far from the wind and wave effective areas, preserving the island's overall structure.

2.4.5 Coral reef modeling

Once the terrain has been generated and deformed by the wind, the system simulates the subsidence of the volcanic island and the growth of coral reefs. These processes reflect the long-term geological evolution of coral reef islands, where the volcanic island gradually sinks (subsides) while coral reefs grow upward to "keep-up" with the sinking landmass.

Subsidence

The subsidence of the island is modeled by scaling the initial height field downward, simulating the effect of the volcanic island slowly sinking into the ocean. The user provides a subsidence rate λ, which represents the proportion by which the island has sunk over time. The subsidence is applied uniformly to the terrain, meaning all points on the island sink by the same factor.

The subsided height field $h_{\text{subsid}}(\mathbf{p})$ is computed by scaling the original height field $h_0(\mathbf{p})$ with the subsidence factor $\lambda \in [0, 1]$:

$$h_{\text{subsid}}(\mathbf{p}) = (1 - \lambda) \cdot h_0(\mathbf{p}) \tag{2.6}$$

This scaling reduces the overall height of the island, simulating how volcanic islands sink over time due to tectonic activity and erosion. The subsidence factor λ is applied uniformly across the terrain, meaning that all points on the island experience the same degree of subsidence, regardless of their original height or location.

Coral reef growth

As the volcanic island subsides, coral reefs grow upward to remain close to the water surface, following the "keep-up" strategy observed in most real-world coral formations. Coral growth is restricted to regions where the depth is within the optimal range for coral development, typically from the water surface to around 30 meters below before being much scarcier.

The coral reef features (reef crest, back reef, and fore reef) are modeled separately from the subsidence process. The system generates a coral feature height field $h_{\text{coral}}(\mathbf{p})$, which remains unaffected by the island's subsidence. This height field ensures that coral regions remain near the water surface, even as the island sinks.

In our model, coral reef growth is entirely independent of the subsided terrain. Even as the volcanic island sinks, coral growth is driven only by the proximity of terrain to the water surface, ensuring that coral features always remain near the surface, irrespective of how much the island subsides.

The coral reef height field is generated using predefined depth values for the various coral regions:

- The reef crest is modeled near the water surface, typically just below sea level,
- The back reef and lagoon are slightly deeper but remain within the range where corals can grow,
- The fore reef slopes downward into the deep ocean, transitioning into the abyss.

The system uses these predefined regions to assign heights to coral reef points based on their proximity to the reef crest. For example, the height of a point in the fore reef is interpolated between the reef crest height and the deeper abyss regions.

Blending the height fields

The final step is to blend the subsided height field $h_{\text{subsid}}(\mathbf{p})$ with the coral feature height field $h_{\text{coral}}(\mathbf{p})$ to produce the final terrain. The goal is to ensure that coral features remain near the water surface while allowing the rest of the island to subside.

To achieve this, the system uses a smooth max function, which smoothly blends the two height fields. The smooth max function ensures that the coral regions dominate where coral growth is present, while the subsided island terrain dominates in other regions. This blending method ensures that the transition between the coral and subsided regions is smooth and visually consistent.

The smooth max function, adapted from Ingo Quilez's smooth min function, is defined as:

$$\text{smax}(a, b, k) = a + \frac{(b - a)}{1 + \exp\left(-k \cdot (b - a)\right)} \tag{2.7}$$

Here, $a = h_{\text{subsid}}(\mathbf{p})$ is the height from the subsided island, $b = h_{\text{coral}}(\mathbf{p})$ is the height from the coral reef feature, and k controls the smoothness of the transition.

This smooth max function guarantees visual continuity by preventing abrupt height differences between the coral regions and the subsided terrain, creating a smooth, gradual transition that mimics the natural blending of coral reefs with deeper areas. The coral feature height field takes precedence where coral can grow, typically in shallow regions. In deeper regions, such as the abyss, the subsided height field naturally dominates, ensuring that the final terrain accurately reflects both subsidence and coral growth processes.

Output

The resulting terrain represents a plausible coral reef island, where the volcanic island has subsided, and coral reefs have grown upward to keep pace with the water level. The smooth blending between the subsided terrain and the coral features ensures a natural transition between regions like the island, lagoon, and coral reefs.

One of the key strengths of this method is its flexibility as the subsidence and coral reef growth processes are modeled independently, allowing for a wide range of configurations. Users can generate plausible island terrains with or without coral features, or apply the coral reef growth simulation to existing height fields from other sources.

2.5 Conditional Generative Adversarial Networks

In this section, we introduce the use of a Conditional Generative Adversarial Network (cGAN), specifically the pix2pix model, to enhance the island generation process by increasing the variety and flexibility of terrains. While the initial procedural algorithm can create numerous island examples, cGAN provides additional flexibility in generating more complex terrain without the rigid constraints of the first algorithm that stem from our initial assumptions.

2.5.1 Introduction to cGAN

A Generative Adversarial Network (GAN) consists of two competing neural networks: a generator that attempts to produce realistic data, and a discriminator that tries to distinguish between real and generated data. In the conditional GAN (cGAN) variant, both the generator and discriminator are conditioned on some input, meaning the generated output is influenced by additional information, such as a labeled map or image.

For island generation, we use the pix2pix model, a specific cGAN designed for image-to-image translation. In this case, the input to the generator is a labeled map which is a 2D image where each pixel is assigned an ID representing a different region of the island (e.g., island body, beach, lagoon, coral reef). The output is an image where each pixel corresponds to the elevation of the terrain at that point, ie. a height field.

2.5.2 Pix2pix model

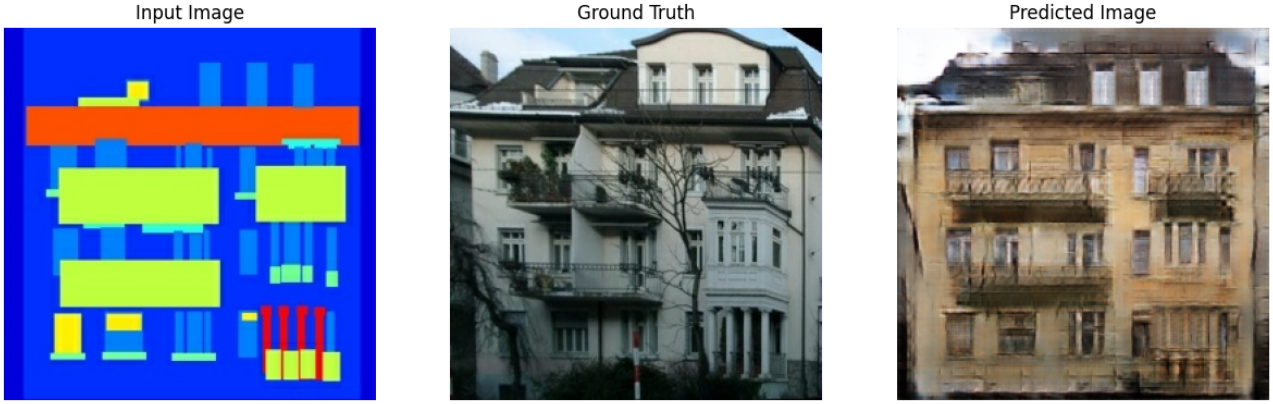


Figure 2.11: Example of application of the pix2pix deep learning model: From a trained dataset of facades composed of real world images associated with their labeled maps, providing a new labeled map outputs a realistic facade image.

[- Bla bla about how the model works...]

2.5.3 cGAN for island generation

The cGAN model was chosen for this task because it allows us to overcome some of the constraints of the initial procedural algorithm. While the first algorithm generates island terrains based on radial symmetry and a limited set of input parameters, cGAN can generate

more complex and varied terrains by learning from a large dataset of examples. Specifically, the use of cGAN addresses the following challenges:

Increased variety: The cGAN model can generate a wide range of island shapes and terrains by learning from the dataset. This allows for the creation of terrains that go beyond the predefined structures of the initial algorithm, introducing more natural variation in island features.

Overcoming constraints: In the initial algorithm, islands are always centered in the image and adhere to a radial symmetry. Using cGAN, we apply data augmentation (translation, scaling, and copy-pasting multiple islands in one sample) to remove these constraints, allowing islands to take more irregular shapes and positions.

Flexibility: Once trained, the cGAN model acts as a black box that generates island terrains based on the labeled input. While this limits user control during the generation process, the model can produce a variety of terrains that still respect the underlying structure of the labeled map. This enables a flexible, rapid generation process without requiring the user to fine-tune parameters manually.

2.5.4 Training

The cGAN model is trained using a dataset of island terrains generated by the initial algorithm. To create this dataset, the algorithm randomly generates top-view shapes for the island's features, adds noise (such as fractional Brownian motion, or fBm), and introduces random wind velocity fields. The profile function remains consistent in determining the relative positions of the island, beach, and lagoon, but the top-view shapes vary through noise and deformation.

Data augmentation

To enhance the variety of the dataset and improve the model's ability to generalize, we apply several data augmentation techniques: - Translation: Since the original algorithm always centers the island, we translate the islands within the image to remove this constraint. This ensures that the cGAN can generate islands in any position within the frame. - Directional Scaling: By scaling the terrain in one direction, we create elongated islands that resemble corridors or archipelagos, adding another layer of diversity to the dataset. - Multiple Islands per Sample: In some cases, we combine multiple islands into a single sample, ensuring they do not overlap. The regions not covered by any island are assigned the abyss ID. Although this approach ensures non-overlapping regions, future work could explore using blending techniques to position islands more closely without the risk of overlap.

All augmentation techniques are applied both to the height field and the labeled map simultaneously to ensure consistency between the input (labeled map) and the output (height field).

2.5.5 Model output

Once trained, the cGAN model generates a height field from a given labeled map. The output is a complete 2D height map representing the terrain's elevation at each point. Although the cGAN works as a black box, the labeled map remains available after inference and retains valuable information for the user.

The labeled map can be used for post-processing tasks, such as:

Texturing: Different regions of the terrain (e.g., beach, lagoon, coral reef) can be textured based on their region ID from the labeled map, allowing for detailed terrain decoration.

Material information: The labeled map can provide cues about the underlying ground material in each region, which can be useful for additional simulations, such as erosion or weathering.

However, it's important to note that the current implementation of the cGAN does not allow for user control during the generation process itself. The model takes the labeled map as input and outputs a height field without any additional parameters that the user can adjust in real-time. While this limits user interaction, the resulting terrains are still varied and flexible thanks to the data diversity learned during training.

2.5.6 Limitations

While the cGAN model provides increased flexibility and variety in island generation, it does come with certain limitations:

Biases from the synthetic dataset: Since the cGAN model is trained entirely on a procedurally generated dataset, it inherits the biases present in the initial algorithm. For example, while the model can break free from the radial symmetry constraint and center positioning, it still relies on the synthetic data's structure and patterns. This can limit the true diversity of the generated terrains, as the cGAN cannot generate terrains that deviate too far from the examples in the training set.

Lack of user control: Another limitation of using cGAN in this context is the lack of real-time user control during terrain generation. While traditional procedural generation methods allow users to tweak parameters (e.g., island size, beach width) during the generation process, the cGAN model operates as a black box, providing no mechanism for direct user interaction beyond the initial labeled map. This reduces the level of customization available to the user.

Data-driven dependence: The quality of the generated terrain depends entirely on the quality and variety of the training dataset. Since the dataset is synthetically generated, any limitations or biases in the initial dataset directly affect the cGAN's output. This dependence on data quality makes it crucial to design a well-augmented and varied dataset to ensure diverse and realistic outputs.

2.6 Conclusion

This work has presented a novel approach to generating coral reef island terrains by combining traditional procedural methods with deep learning techniques. We first developed a procedural generation algorithm capable of creating a wide variety of island terrains through a combination of top-view and profile-view sketches, wind deformation, and subsidence and coral reef growth simulation. By applying these methods, we were able to produce realistic terrains based on geological processes, capturing key features of coral reef islands such as beaches, lagoons, and coral reefs.

To further enhance flexibility and realism in the generation process, we incorporated a Conditional Generative Adversarial Network (cGAN), using the pix2pix model to generate height maps from labeled maps of island features. The cGAN model allowed us to overcome some of the constraints inherent in the procedural algorithm, such as radial symmetry and fixed island positioning. With data augmentation techniques, we were able to train the cGAN on a synthetic dataset, generating varied and realistic island terrains.

2.6.1 Advantages of the approach

One of the main strengths of this approach is its ability to produce a wide variety of island terrains, even in the absence of real-world data. The procedural generation methods allow for high flexibility in designing both the shape and features of the island, while the use of cGAN enables further refinement and the generation of terrains that are not bound by the original constraints of the procedural model. By combining these two methods, we leverage the advantages of both: the structured control of procedural techniques and the pattern-learning capabilities of deep learning.

A key advantage of this approach is the retention of semantic information about the terrain throughout the generation process. The labeled map, which serves as the input to the cGAN, can also be used after terrain generation to provide a detailed representation of the different regions of the island (such as the beach, lagoon, coral reef, and island body). This labeled map can guide post-processing operations, such as applying different textures based on terrain features or adding other environmental elements like vegetation. The preservation of semantic information provides a useful connection to the next stage of terrain manipulation, making the process more versatile and adaptable to different use cases.

Furthermore, the use of an out-of-the-box cGAN model highlights the feasibility of employing existing neural network architectures with minimal modifications in the field of procedural generation. This is particularly important in domains where real-world data is scarce, such as coral reef islands, allowing synthetic data to be effectively used for training purposes.

2.6.2 Limitations

While this approach brings significant advantages, there are also some limitations to consider. The reliance on a synthetic dataset means that the cGAN inherits the biases and limitations of the original procedural algorithm. This could limit the true diversity of the terrains that the model can generate, as the output is confined by the patterns present in the training data. Additionally, the cGAN model functions as a black box, offering limited user control over the generation process once the model has been trained. This contrasts with traditional procedural methods, which typically allow for real-time tweaking of parameters.

2.6.3 Future works

There are several directions for future research and improvements. One promising avenue is to incorporate the wind velocity field more directly into the cGAN training process, potentially as an additional input condition. This would allow the model to better capture wind-driven terrain features such as cliffs or other deformations influenced by wind patterns.

Another area for exploration is improving user interaction during the terrain generation process. While the current model allows for rapid terrain generation, adding more options for users to interact with the cGAN, such as tweaking parameters like wind strength or island size, could enhance the flexibility of the system.

Finally, further improvements could be made to the synthetic dataset. Incorporating more complex geological processes, such as wave erosion or tidal influences, could lead to even more realistic terrains. Additionally, refining the way islands are blended in multi-island samples, or adding more diverse input conditions (e.g., different geological settings), could help the model generalize better and produce more varied and dynamic landscapes.

One possible future improvement could involve incorporating the wind velocity field into the cGAN training process. While the labeled map is the only input used in the current implementation, the wind field could be added as an additional condition. This would be especially useful if the initial algorithm were augmented to include wind-driven features, such as cliffs or specific terrain deformations influenced by wind patterns. Adding the wind field as an input could help the cGAN generate more realistic terrains that better reflect the influence of wind on the landscape.

Additionally, further development could explore improving how multiple islands are combined in a single sample. For example, using blending techniques to handle overlapping regions could allow islands to be positioned closer together, enabling the generation of more complex archipelagos without sacrificing the integrity of the height field.

Many other neural networks models could be exploited to increase the possibilities, such as models with style transfer functionalities (**Gatys2015**; **Zhu2020**) in order to change the overall aspect of a terrain (**Perche2023a**; **Perche2023b**), use text-to-images models (**Rombach2022**; **Radford2021**) to generate height fields from a verbal prompt, or super-resolution models (**Dong2015**) to increase the definition of details in the final output (Guérin, Digne, Galin, and Peytavie, 2016).

CHAPTER 3

Semantic terrain representation

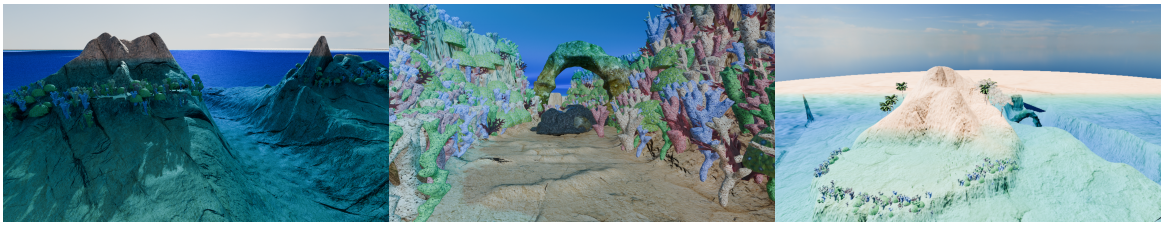


Figure 3.1: Our method can produce different scenes including coral reef islands and canyons at multiple scales using environmental objects to represent terrain features.

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 "environmental objects", 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.

Contents

Back to summary

3.1 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 (more rarely) polyhedrons. These symbols are a simplification of the content of an environment, or an abstraction of the 3D nature of the terrain features as we can see in the examples in Figure 3.2. 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 artificial reefs on coastal erosion, or understand more clearly the interactions inside an ecosystem. Oceanologists may also deduce, using the same strategy, the formation of canyons and fans from old river systems. [A REDIRE]

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.

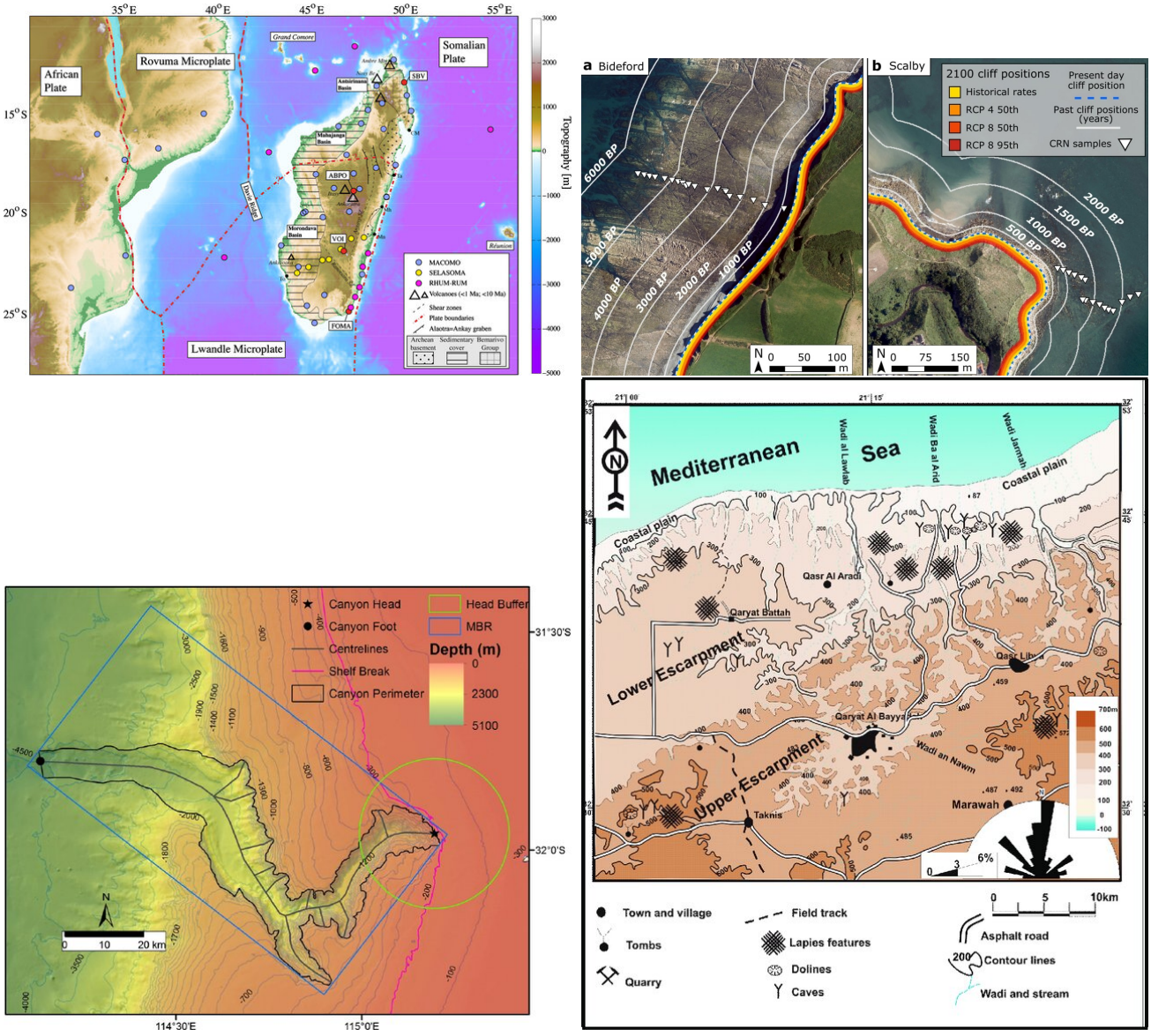


Figure 3.2: Four examples of topographic maps used in earth science. From left to right and top to bottom: Sedimentary distribution other Madagascar island (Pratt et al., 2017), evolution of coastlines at Bideford, UK and Scalby, UK over the last 6000 years and 2000 years respectively (BP = Before Present) (Shadrick et al., 2022), localization of key parameters of Perth Submarine Canyon, Autralia (Huang, Nichol, et al., 2014), and geological features distribution of karstic landscape at Qasr Lybia, Libya (Amawy and Muftah, 2009)

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 may 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.

3.2 Related works

Procedural terrain generation has been heavily studied for the last 40 years (Galin, Guérin, 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 (Kapp et al., 2020; Brosz et al., 2007), while the rise of deep learning technologies gave higher control to the user through sketches (Guérin, Digne, Galin, Peytavie, et al., 2017; Talgorn and Belhadj, 2018).

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 (Smelik, Tutenel, Bidarra, and Benes, 2014). 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.

While stochastic noise can be sufficient to model coarsely the ocean floor (Mareschal, 1989), this process won't cover areas with the biggest biomass, near shallower waters such as near coasts and islands. These areas, that represent a very small portion of the oceanic surface, are much more complex as many interactions between biolife, air and water are in action.

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' growth simulation (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.

Recent works have been presenting "feature tools" to correct landscapes from unaccurate DEM (2.5D) using vector-based features that modifies the geometry of the ground and water bodies to fit thir respective 2D satellite images, by explicitely defining the position of natural features like rivers and mountains (Ketabchi et al., 2016). Visible 3D features like vegetation and buildings can also be added in the final result as meshes affected to a single point. This leads to sketch-based applications where features are represented like topographic maps. This solution allows for the manipulation of an existing terrain, guided by a real-world satellite image, but lacks the possibility to completely generate an unseen landscape or for the terrain features to interact between them. Feature tools have been proposed to generate terrains from scratch (Smelik, Tutenel, Kraker, and Bidarra, 2010), but usually require to define in advance the interactions between each feature like a automatically displaying a bridge when a river crosses a road.

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; Guérin, Digne, Galin, and Peytavie, 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, they called environmental objects, 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, parallely Wejchert and Haumann (Wejchert and Haumann, 1991) and Sims (Sims, 1990) 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

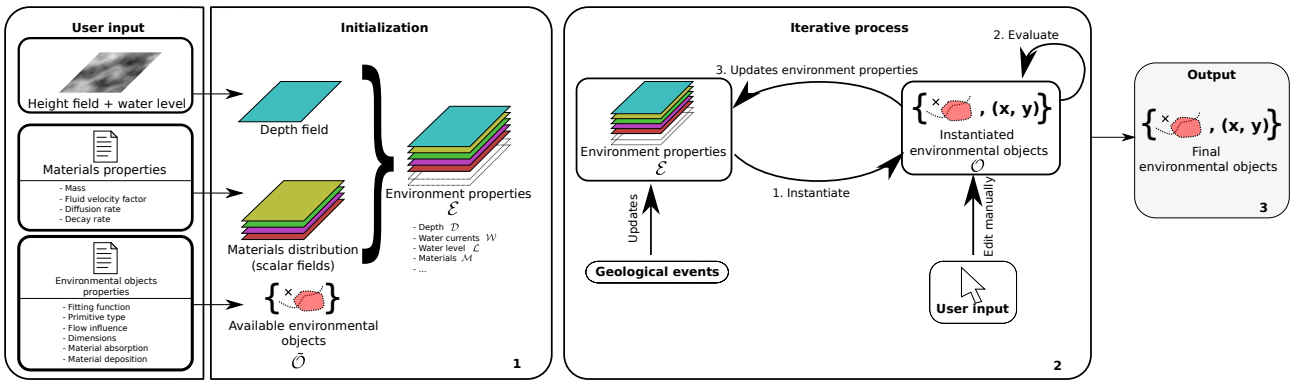


Figure 3.3: Overview of the pipeline of the method. The user provides as input an initial height field and sets the water level, as well as a definition of the environmental materials properties and environmental objects properties that will be used in the iterative process. These inputs are initialized as an initial set of environmental objects and scalar fields that represents the environmental attributes. In the iterative loop, new environmental objects are instantiated using the current state of the environment at their optimal position. The existing environmental objects 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 environmental objects. The result of the whole process is a set of environmental objects which is a sparse representation of the features of the scene.

low computation effort and preserving a high user control.

3.3 Description of the method

In this section, we present the pipeline and processes that underpin our method. Our approach introduces the concept of environmental objects, simplified terrain features that interact with their surroundings to simulate complex ecosystem dynamics. These environmental objects, 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 environmental objects are set up; an iterative generation process, where these environmental objects 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.

3.3.1 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 environmental objects $\widetilde{\mathcal{O}}$, 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 3.3.2). A target list of environmental objects $\hat{\mathcal{O}}$ 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 environmental objects 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 3.3.3).

The definition of environmental objects' 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 (**??**).

The generation phase starts with an initial set of environmental objects 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 environmental objects then the update of the environment. This loop is repeated until the user is satisfied with the look of his environment or following rules like a number of features threshold or a targeted list of environmental objects as described in the layout planner defined in Tutenel et al., *Rule-based layout solving and its application to procedural interior generation* (2009) .

Instantiation

At each iteration, new environmental objects 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 3.4). All environmental objects are evaluating their state analytically using a fitness function and a skeleton fitting function provided as input (Section 3.4).

Environment update

Once the instantiation step is done, the environmental attributes are updated by each environmental object through environmental modifiers, which depose and absorb some of the environmental materials $\mathcal{M}$ (Section 3.5) while modifying the water currents $\mathcal{W}$ (Section 3.5.3) 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 environmental objects (Section 3.6.1) and perturb the generation process by planning geomorphic events that have impacts on the environmental attributes (Section 3.6.2).

Output

The output of our system is a set of environmental objects disposed in the plane. We do not provide the 3D representation of the environmental objects 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.

3.3.2 Environmental objects $\mathcal{O}$

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 environmental object in this work.

Each environmental object 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 environmental objects as point-based, curve-based or region-based, respectively. These environmental objects 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 environmental object 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.

3.3.3 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 environmental object to interact with any other one. Each of the environmental object have a local impact on the environmental attributes without knowledge

of neighboring environmental objects. This modification of the environmental attributes are presented as the effect of environmental modifiers defined for each environmental object.

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 environmental objects, rather than as something directly observable in the scene.

Environmental modifiers $\mathcal{E}^+$

The environment determine if a environmental object does belong at a certain position. When a environmental object 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 $\mathcal{M}^+$.

Environmental objects 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 environmental object. 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 environmental object 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 environmental object.

Each environmental object 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 environmental objects.

The alteration of a scalar environmental attribute is done by adding or removing some amount around the skeleton of the environmental object 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.

3.4 Placement of environmental objects in an environment

At each iteration of our algorithm, we want our environmental objects 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 environmental object in order to satisfy the users wishes, while conserving the plausibility of the scene. Rather than "forming" these environmental objects, 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 environmental object that may be spawn in the terrain. The fitness function defined $\omega : \mathcal{E} \mapsto \mathbb{R}$ provides a score indicating how well the environmental object 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 environmental object 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 environmental object. We know that each environmental object will modify the environment surrounding, which may make previously instantiated environmental objects 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 environmental object at a certain position at each iteration and evaluate the stability of the environment, repreating this operation while varying slightly the position of the environmental objects or the type of environmental object 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 environmental objects 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 environmental object 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 environmental object 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.

3.4.1 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 environmental object 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 environmental objects, 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 environmental object being considered.

Different types of environmental objects 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 environmental object, ensuring that the generated terrain remains coherent and realistic.

3.4.2 Skeleton fitting function Γ

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

Point-based skeleton

The spawning position of a punctual environmental object 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 environmental objects. 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

The skeleton of a curve-based environmental object is determined by the shape that fits the most given the environment it is added to. Using a modified version the Active Contours algorithm (Kass et al., 1988), we can minimize the energy E for the parametric curve C given

$E(C) = E_{\text{internal}} + E_{\text{external}} + E_{\text{shape}} + E_{\text{gradient}}$ with

$$E_{\text{internal}} = \alpha_{\text{i}} \left(\int_C \left\| C'(s) \right\|^2 ds + \int_C \left\| C''(s) \right\|^2 ds \right) \tag{3.1}$$

$$E_{\text{external}} = \alpha_{\text{e}} \int_C \Gamma(C(s))\, ds \tag{3.2}$$

$$E_{\text{shape}} = \alpha_{\text{s}} \left(L - \int_C ds \right)^2 \tag{3.3}$$

$$E_{\text{gradient}} = \alpha_{\text{g}} \int_C \langle C'(s), \Delta\Gamma(C(s)) \rangle\, ds \tag{3.4}$$

In this configuration, E_{internal} induce a smooth continuity of the curve by reducing the spacing of each point while reducing the curvature. Another energy, E_{external} integrate the skeleton fitting function over the curve, often seen as an attractor of the points, that tries to descent the gradient to find local minima. At the same time, E_{shape} apply constraints on the curve shape, which, in this case, is to target a specific length L. As such, the curve search for an optimized shape given constraints in E_{shape} for minimizing E_{external}. We introduced a new term E_{gradient} in the energy computation that push the points of the curve in the direction of the slope of the skeleton fitting function, also providing an orientation for the curve.

If a steep coast can be found where the terrain slope is important near the water level, we can define $\Gamma = |\mathcal{H} + 1| \,/\, \|\Delta\mathcal{H}\|$, but no orientation is needed, thus we set $\alpha_{\text{g}} = 0$. In this case, the curve will follow a path at the water level and spread its extremities over areas with a steep slope. A river may also be symbolized as a parametric curve, but we need to add information about the direction and magnitude of the slope. As such, we can use $\Gamma = \mathcal{H}$ which forces the direction of the curve to fit with the terrain slope. The introduction of the gradient component provides also an orientation to shapes.

Internal energy

The internal energy, already introduced in Kass et al., *Snakes: Active contour models* (1988) is composed of two components imposing penalities on the local properties of the points of the curve. The first derivative forces the points along the curve to be evenly spaced by minimizing the curve tension, while the second derivative restrict it from forming sharp corners. As our aim is to represent natural elements, we rarely find sharp elements and thus, keep the original definition from the Snake formulation.

External energy

The external energy is also present in the original work. Using an external scalar field, each point of the curve is forced to follow the steepest slope of the field. The external field can be seen as an attractor for the curve.

Shape energy

We introduced the shape energy, an energy defined on the whole curve to apply constraints on its final shape. As many natural features have a given dimension, we may add a constraint on the lenght of the skeleton.

Gradient energy

In our application, having information about the orientation of environmental objects may be essential. For this purpose, we introduced a gradient energy component in the formulation. The equation impose that the direction of the curve at any point should be directed towards the gradient of the scalar field. As the external energy pushes points toward the lowest point

of the scalar field, the gradient field restrict the gradient descent for the global curve into a specific way, which may feel more natural.

During the optimization process, the gradient of the scalar field is already evaluated by the external energy optimization, so the addition of the gradient component is almost free.

Region-based skeleton

Region-based environmental objects follows the same process than curve-based environmental objects to define their skeleton, at the exception of the gradient energy E_{gradient} which have null value on closed shapes.

The resulting energy E to minimize for a closed region whose borders are defined by the curve C can then be expressed as $E(C) = E_{\text{internal}} + E_{\text{external}} + E_{\text{shape}}$.

The internal energy is expressed identically as for curve-based environmental objects. The extenal energy however, has to be modified to take into account the interior of the region instead of only the borders. We will use the idea of the Chan-Vese algorithm, differentiating the energy value for the inside Ω and the borders C of the region (Chan and Vese, 2001; Getreuer, 2012).

By adding a factor for the inside λ_1 and a factor for the borders λ_2, we can add weight depending on the required optimization. We can then define the external energy component E_{external} as:

$$E_{\text{external}} = \lambda_1 \int_{\Omega} \Gamma(s)\, ds + \lambda_2 \int_{C} \Gamma(C(s))\, ds \tag{3.5}$$

We see that the definition of the energy formulation of the curve-based environmental objects is a specialization of the region-based, with $\lambda_1 = 0$ and $\lambda_2 = \alpha_{\text{e}}$

The use of external forces on the skeleton can be useful as some landscape features are easier to define by what they contain, while other by what they separate. For example, a lagoon is formed by coral reefs surrounding it, while a forest is formed by the climatic conditions inside it that are propice for trees. A lagoon may then be defined with $\lambda_1 = 0$ and $\Gamma_{\text{lagoon}} = -\mathcal{M}_{\text{coral limestone}}$ while a forest sets $\lambda_2 = 0$ and $\Gamma_{\text{forest}} = -\mathcal{M}_{\text{humidity}} + \mathcal{M}_{\text{shade}} + \left|\mathcal{M}_{temperature} - 10\right|^2$.

Finally, the shape constraint energy E_{shape} can target an area A, for example.

$$E_{\text{shape}} = \left(A - \int_{\Omega} ds\right)^2 \tag{3.6}$$

In our implementation, each environmental object's skeleton is a connected component as we define the boundaries by the connected curve C, but can be convex or concave. An infinite penality is added for the curve self-intersection.

At each iteration, environmental objects are interrogated to verify if they are still fitted to belong at their position. We first check that the fitness is above a given threshold $\omega > T_{\omega}$. If not, we remove the environmental object from the scene. On the other case, we can improve the shape of the skeleton by minimizing again their skeleton fitting function. As the number of environmental objects become important in the scene, the reajustment time for the skeletons becomes important. For environmental objects that have been present for multiple iterations, we consider less iterations and an increased convergence threshold up to a point where the features are static.

3.5 Environmental modifiers

Each environmental object once present in the scene has an impact on the environment $\mathcal{E}$ through their modifiers $\mathcal{E}^+$. We list three types of influence, which are the environmental material modifiers $\mathcal{M}^+$, the height modifiers $\mathcal{H}^+$ and the water modifiers $\mathcal{W}^+$. These modifiers are direct impact and can be computed as

$$\mathcal{E}^* = \mathcal{E} + \sum_{o \in \mathcal{O}} \mathcal{E}_o^+$$

3.5.1 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 environmental object in the terrain is a source and a sink of environmental materials. It is the main mean of communication between environmental objects 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 environmental object and $\gamma(t) \in [0, 1]$ a factor related with the current state of the environmental object, which state that more material will be displaced when the environmental object is fully formed than when it was just spawn:

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

The deposition and absorption around an environmental object 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

$$\frac{\partial\mathcal{M}}{\partial t}\Phi\nabla\mathcal{M} = D\nabla^2\mathcal{M} - k\mathcal{M} \tag{3.7}$$

We solve Equation (3.7) numerically using Euler integration

$$\begin{aligned}\mathcal{M}(\mathbf{p}, t + dt) = \mathcal{M}(\mathbf{p}, t) + dt(D\nabla^2\mathcal{M}(\mathbf{p}, t) - k\mathcal{M}(\mathbf{p}, t) \\ - \Phi(\mathbf{p}, t)\nabla\mathcal{M}(\mathbf{p}, t))\end{aligned} \tag{3.8}$$

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.

3.5.2 Height modifiers

The computation of the height $\mathcal{H}(\mathbf{p})$ is done using the coarse height function of each environmental object affecting a point $\mathbf{p}$. We can then simplify the shape of a mountain to a cone, a reef as a curved cylinder or a coral boulder as a sphere. As we consider surface height and not surface volume, we use the top-view height field projection, which ignore overhangs and cavities.

The coarse height function is an implicit height field that is used for estimating the shape of the environmental object while providing a hint of the surface normal at each point. Using implicit surfaces allows us to evaluate the altitude and the slope at any point analytically, without requiring the computation of the whole field. The analytical solution induce the multi-scale aspect of the method.

We divide the coarse height functions in three categories:

- $\mathcal{G}$: environmental objects whose shape is defined from an absolute zero of the terrain,
- $\mathcal{A}$: environmental objects whose shape is defined using a notion of altitude, typical of coral-related elements as the depth from the water surface is prevalent,
- $\mathcal{F}$: environmental objects that are defined at the terrain surface. They represent objects that can be seen as bumps or carves from an aerial view.

We compute the depth change given for each category:

$$\mathcal{H}^+_{\mathcal{G}}(\mathbf{p}) = \max_{o \in \mathcal{G}} \mathcal{H}^+_o(\mathbf{p})$$
$$\mathcal{H}^+_{\mathcal{A}}(\mathbf{p}) = \min_{o \in \mathcal{A}} \mathcal{H}^+_o(\mathbf{p})$$
$$\mathcal{H}^+_{\mathcal{F}}(\mathbf{p}) = \sum_{o \in \mathcal{F}} \mathcal{H}^+_o(\mathbf{p})$$

The final altitude is computed as

$$\mathcal{H}^+ = \max(\mathcal{H}^+_{\mathcal{G}}, \mathcal{H}^+_{\mathcal{A}}) + \mathcal{H}^+_{\mathcal{F}} \tag{3.9}$$

3.5.3 Influence on water currents

We define our water currents as a vector field defined as

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

With $\mathcal{W}_{\text{user}}$ a user-defined vector field, $\mathcal{W}_{\text{simulation}}$ an analytical solution directly inspired by a wind flow simulation (Paris, Peytavie, et al., 2019), and $\mathcal{W}^+_{\text{objects}}$ the water flow alteration computed from the environmental objects. The component $\mathcal{W}_{\text{simulation}}$ is influenced by terrain surface level. Starting with an initial flow direction a, the vector field is adjusted by applying a warp influenced by the terrain gradient at various scales:

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

Here, v is calculated as $v = a\,(1 + k_w\,|\mathcal{H}(\mathbf{p})|)$, where $|\mathcal{H}(\mathbf{p})|$ represents the distance to water level at point $\mathbf{p}$, and k_w is a scaling factor that accounts for Venturi effects. The term $\Phi_i \cdot v$ denotes the warping process at scale i, with c_i as the associated coefficient, defined as follows:

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

In this formulation, k_i serves as the deviation coefficient, α represents the gradient of the smoothed terrain, and $\nabla\widetilde{\mathcal{H}_i}^{\perp}(\mathbf{p})$ is the vector orthogonal to the smoothed terrain. Consistent with the original paper, two scaling levels were employed ($n = 2$) using Gaussian kernels with radii of 200 m and 50 m. The weights were set to 0.8 and 0.2, with corresponding deviation coefficients $k_0 = 30$ and $k_1 = 5$.

$\mathcal{W}^{+}_{\text{objects}}$ is a deformation field defined as the accumulation of flow primitives (Wejchert and Haumann, 1991). Kelvinlets are applied on each environmental objects 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

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

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 $\mathbf{q} = \mathbf{p}_C^*$ with $\mathbf{p}_C^*$ 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:

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

3.5.4 Environment stability

Environmental objects 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 environmental objects 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.

Figure 3.4: 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*).

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 environmental object can alter the balance, leading to a new phase of interaction and stabilization.

3.6 User interactions

The user can guide the generation process. The use of simple shapes as environmental objects facilitate the edition of the simulation, as we can interactively add, remove or modify environmental objects, 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 environmental objects are instantaneous, as a the geomorphic events are active on a given duration.

3.6.1 Direct interactions with the environmental objects

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

Translating an environmental object is trivial, we simply requires to evaluate the state of the environmental objects at a translated position. The deformation of environmental objects can be applied on curve and region environmental objects by updating the control points of the skeleton and recomputing the resulting implicit surfaces. The evaluation positions used for region environmental objects 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 environmental object, the environmental attributes may change, which can result in the destruction of the now incompatible environment objects in the scene (Figure 3.4).

As long as a non-zero fitness function is defined in the terrain, new environmental objects 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 environmental object 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 3.8 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 environmental objects 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 3.5.3).

3.6.2 Indirect interaction with environmental objects

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 geomorphic 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 environmental objects 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 environmental object that is sensible to the depth will be impacted automatically, that may be causing death (Figure 3.5). 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 3.6). 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

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

with $G(x)$ the Gaussian function

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

Storms are factors of the geomorphology of coral reefs (Vila-Concejo and Kench, 2016; Oron et al., 2023) and coasts (Domínguez et al., 2005; Cowart et al., 2010). Due to the extreme

Figure 3.5: 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.

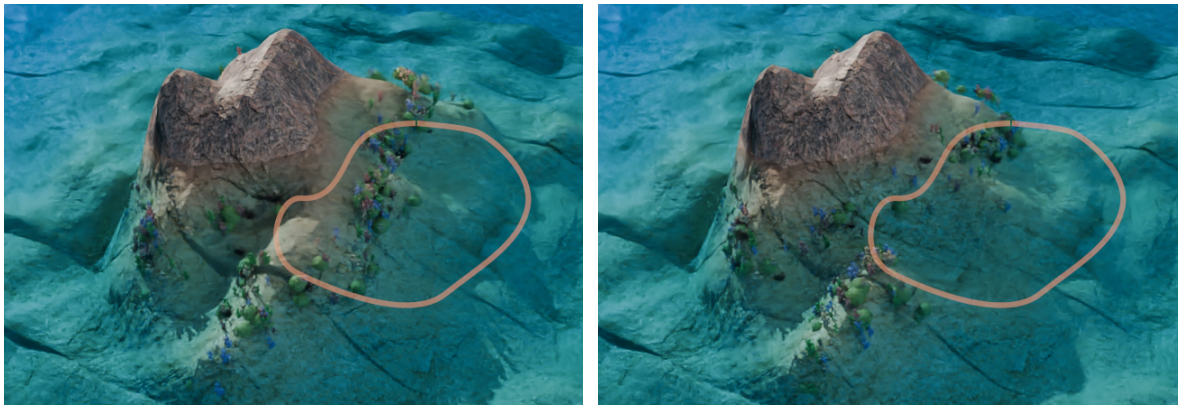


Figure 3.6: Simulating subsidence on a part of the terrain (brown area) cause the depth value to change locally, resulting in the death of coral objects that find themselves too deep to survive. Here two subsidence geomorphic events are triggered in parallel.

Figure 3.7: The result of a storm localized on one side of the island (red area) modifies the result of the evaluation of environmental objects around its epicenter for a short period of time. Most of the coral objects died from the geomorphic event, except few environmental objects less sensible to water currents strength.

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 3.7). The computation of water currents are then computed as

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

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, ...

3.7 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 3.1) after what we focused the generation process in a canyon (Figure 3.9), then a small-scale visualization of coral colonies (Figure 3.8). In the examples, we rendered the environmental objects as a implicit

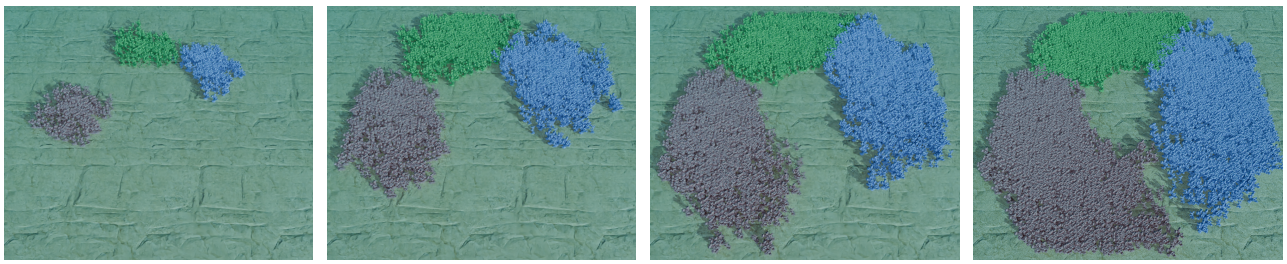


Figure 3.8: Three colonies of coral (red, blue, green) restricted to an annulus the middle section of the terrain fighting for the space.

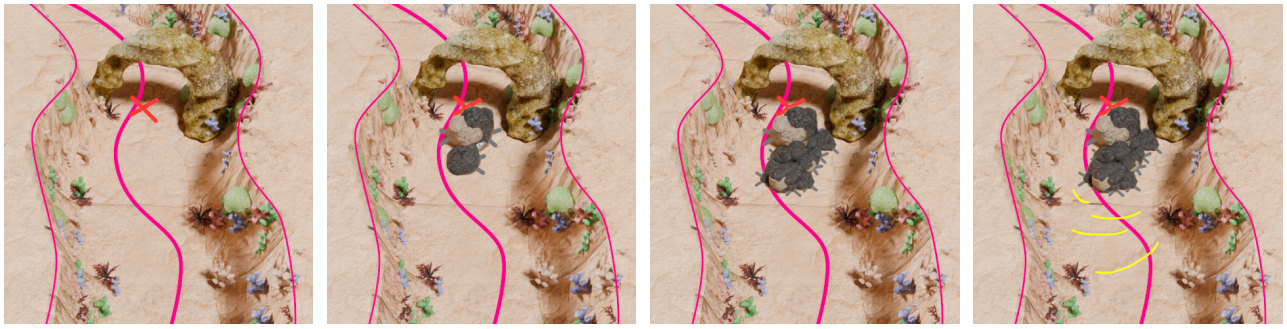


Figure 3.9: Evolution of a canyon scene at different iterations of the simulation. The apparition of an arch causes the spawning of rocks, pebbles, and finally some deposition of sand at the bottom of the canyon, spawning ripples.

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 environmental object "Arch" with a fitness function $\omega_{arch}(\mathbf{p}) = 5 - d(canyon - \mathbf{p}) * \|\mathcal{W}(\mathbf{p})\|$, an environmental object "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 3.9 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 3.8 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 environmental objects. 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 environmental objects 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 environmental objects.

3.8 Conclusion

We have proposed a method to generate terrains procedurally using sparse representations. This representation, the environmental objects, enables to introduce expert knowledge by the mean of the fitness functions that rule the environmental objects 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 environmental objects 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 environmental objects allows the user to guide the generation through geomorphic events. Our method enables each environmental object 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 environmental object'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 3.10: A simple coral reef island is generated using an island, a lagoon, reefs coral polyps, beaches, trees and algae environmental objects. Trees appear on beaches and algae grow in the lagoon's sand.

CHAPTER 4

Volumetric terrain modeling

Abstract

Once the global shape of the environment is defined by the designer, the modeling of the surface may begin. Modeling a terrain allows us to visualize the surface of the terrain and see concretely the result of the generation process. In the modeling process of landscape features, the method used for each of the element may be completely different as the natural phenomena forming or creating them may have very different natures. With procedural generation, the balance between speed, realism and control have to be stroke. For this reason, the same geological feature may be simulated using completely different algorithms.

The modeling part of this thesis is divided in two chapters, each of them focusing on a different landscape feature. In Chapter 2, we will propose a novel algorithm for the modeling of coral reef islands taking advantage of the recent advances in deep learning. We will present our method for generating a large dataset of examplar and the training of a Generative Adversarial Network (GAN).

Secondly in Chapter D, we describe a procedural algorithm for the modeling of karst networks. In this method, we define the path of the cavities, focused toward user control.

Contents

Back to summary

4.1 Introduction

[THIS MIGHT BE THE ABSTRACT OF THE PART]

The output of the semantic terrain generation is a set of environmental objects containing a skeleton defined as points, curves or regions. From this very coarse representation, we want to create the 3D geometry to visualize the real terrain. This parametric geometry representation of the skeletons pushes us to use the implicit surface representation to model it. We extended this idea to work with implicit volumes with material information.

Working with volumes instead of surfaces allows to keep the information of which terrain features occupies which points in space. Adding to this the notion of material information has many advantages: first, we can render different textures procedurally on the ground, the features and even the inside if needed. Secondly, this information can be used in post-processes like the erosion simulation step, where this information enables to weather more or less some parts, or add effects at the interface between multiple materials. Finally, including "invisible" materials like air or water allows to represent 3D objects like tunnels the same way as solid objects.

4.2 Implicit terrains with materials

Material information in the terrain ground provides useful hints for the generation process. Recent works in terrain generation make use of construction trees to aggregate terrain features in a landscape. Construction trees allow to store the sparsely the features for a very low memory cost. While the implicit modeling has been studied for a long time now (Turk et al., 2001), only recently has the . In 3D modeling, the model of the BlobTree(Schmidt et al., 2006) has been often used. More studies about blending functions (Barthe et al., 2004; Bernhardt et al., 2010; Groot et al., 2014; Vaillant et al., 2013; Angles et al., 2017) enabled to find new ways to smoothly integrate the different nodes of the construction trees. In terrain generation, the use of construction trees have been used in 2.5D for adding and blending features together in order to represent a large scene (Génevaux et al., 2015; Guérin, Galin, et al., 2016), but also in 3D to represent smaller features (Paris, Galin, et al., 2021). However,

in these methods, the material information is lost and we consider that everything that is defined by the construction tree is composed of a single ground type.

We extended the idea of construction trees for 3D terrains by extending the operator nodes with positional information. While the problem of material definition inside overlapping objects remains an open problem, we propose solutions to tackle this problematic.

4.2.1 Tree definition

- Tree structure
** Leaves
*** Implicit volumes
*** Material information
*** Bounded support
** Nodes
*** Not always unary or binary
*** Blending functions
*** Positional information
** Evaluation at one point
*** From leaves to trunk
- ...

4.3 Primitives

The leaves of the construction tree are implicit primitives. An implicit primitive is a scalar function $f : \mathbb{R}^3 \mapsto \mathbb{R}$ for which we typically consider the surface at any point $\mathbf{p}$ satisfying $f(\mathbf{p}) = 0$. We consider that we are inside the volume for $f(\mathbf{p}) < 0$ and outside for $f(\mathbf{p}) > 0$. This definition include many possible functions like voxel grids (explicit function) or implicit volumes.

We use the primitives to represent the presence of a feature at a given position, and as such we limited the domain of the function to be $f : \mathbb{R}^3 \mapsto [0, 1]$ where a value closer to 1 is the presence of the given feature and 0 its absence. A common mapping from the "classic" function to the restrained function may look as $f_{\text{restrained}} = \text{sigmoid}(-f_{\text{classic}})$ with the sigmoid function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-t(x-x_0)}} \tag{4.1}$$

with t being the logistic growth rate and x_0 an offset of the function, the x value of the function's midpoint. We typically set $x_0 = 0$ to center symmetry point of the function at $x = 0$. With this definition of the scalar function, the surface previously set at $f_{\text{classic}} = 0$ is transfered to $f_{\text{restrained}} = 0.5$. Keeping our values between 0 and 1 allows us to use more tools as to manipulate the content of our construction trees. By using the logit function

$$\text{logit}(x) = \frac{1}{t} \ln\left(\frac{x}{1-x}\right) + x_0 \tag{4.2}$$

we can get back to the classic implicit formulations. As the exponential function quickly grow, we usually consider $\text{sigmoid}(x) = 1$ for $x \geq \frac{6}{t}$ and $\text{sigmoid}(x) = 0$ for $x \leq -\frac{6}{t}$.

We include an extra information in the node: the composing material. Typically, we represent the material as an index to a set of predefined materials like sand, rock, soil, water, etc... In our representation each primitive is associated to a unique material.

4.3.1 Scalar functions

The scalar functions associated to primitives are defined as $f : \mathbb{R}^3 \mapsto \mathbb{R}$. We usually associate points and curves to the function in order to define parametric volumes.

Examples of scalar functions may include:

Spheres: the parametric equation of the sphere is the most common model, using its center c and radius r. $f(\mathbf{p}) = (c - \mathbf{p})^2 - r^2$.

Tubes: using a parametric curve C and a radius r, we can define tubes, that can represent arches or tunnels, depending on the material associated. $f(\mathbf{p}) = (\mathbf{p}_C^* - \mathbf{p})^2 - r^2$ with $\mathbf{p}_C^*$ defined as the closest point of the curve from $\mathbf{p}$.

Scalar functions are not limited to simple equations. The tunnels digged in Chapter D, for example, use the implicit swept volume formulation (Schroeder et al., 1994), providing a parametric path and a parametric 2D shape to carve the ground. The later is also parametrized to evolve along the curve path in order to have an entry shape and an output shape that can be different.

4.3.2 Material information

The material information given for a primitive inform the evaluator what type of material should be present at any point $\mathbf{p}$ that satisfy $f(\mathbf{p}) < 0$.

Now we also like to define the density of this material inside the primitive, such that in the case where two or more primitives are present in a single point, we can find a unique final material $\mathcal{M}$. In our work we considered the density function $\sigma : \mathbb{R}^3 \mapsto \mathbb{R}$ to be completely related to the scalar function f such as $\sigma = -f$. While not realistic, this default value is sufficient for most cases.

4.4 Operators

In the construction tree structure, nodes represent operators. Unary operators are effective to translate, rotate or scale their child node, but many other manipulations are useful. On the other hand, n-ary nodes, containing two or more child nodes, aggregate their children like CSG trees with their union, soustraction and difference operations, but in a smooth manner, that we call blending.

4.4.1 Unary nodes

An unary node has a unique child. They are used to apply transformations on their child, whever it is an implicit primitive or another node. In CSG, we often use unary operators to apply linear transformation on an object, such as a translation, a rotation or a scaling operation.

Other non-linear transformations may be applied on the child as twisting and warping operations. The twist, usually defined as

$$T(x, y, z) = (x\cos(\theta(z)) - y\sin(\theta(z)), x\sin(\theta(z)) + y\cos(\theta(z)), z) \tag{4.3}$$

with $\theta(z)$ a scalar function defining the rotation strength function of z, resulting in a helice shape, while the warping operator would be defined as

$$W(x, y, z) = (x, y, z) + F(x, y, z) \tag{4.4}$$

with F a vector field that define the warp direction and strength.

Many other operators may be proposed, such as a noise function $f(\mathbf{p}) = f_A(\mathbf{p}) + \eta(\mathbf{p})$.

4.4.2 n-ary nodes

Our scenes may contain a large amount of elements and as such, a binary construction tree representation can become quickly deep and not ideal. We then focused the construction of n-ary trees, where each node can contain an undefined number n of children with $n \geq 2$.

Blending functions

The primitive functions are defined on $f_{\text{primitive}} : \mathbb{R}^3 \mapsto [0,1]$ and we want to keep the result of our blending functions in the same domain $f_{\text{blending}} : [0,1]^2 \mapsto [0,1]$, providing us possible simplifications on the blending functions, which is a good thing as the inclusion of the materials, on the other hand, adds complexities.

First, we need to take into account that some materials are "solid" (sand, rock) and some are "invisible" (water, air). In the CSG paradigm, adding a solid and an invisible object would be seen as the substraction operation. In our work, we consider

- Symmetric blending:
** Analogy easy with the 2.5D version
** With the form $f(\mathbf{p}) = \sqrt[n]{f_A(\mathbf{p})^n + f_B(\mathbf{p})^n}$.
** We use the same function to determine the amount of each material σ_A and σ_B.
** But now, we need to define which material is present where. So nodes contains a vector of materials, such that multiple materials can be defined as each point. This multiple materials information is kept, we will see different solutions to render or collapse this vector in the next section.

- Asymmetric blending:
** A bit more complex than the 2.5D version.
** Replacing:
*** [FIND FUNCTION]
** One-way blending:
*** [FIND FUNCTION]
- ...

Placement functions

- Fixed = "max" in 3D
** A "free" object, or not dependent on environment for example
In this case, we can evaluate intuitively $G(f_A(\mathbf{p}), f_B(\mathbf{p}))$ - Stacking = addition in 3D
** "Liquid" positionning based on object A's surface
The equation for evaluating a position $\mathbf{p}$ of the function f_B stacked over f_A becomes $G(f_A(\mathbf{p}), f_B(\mathbf{p} - \begin{bmatrix}0 & 0 & \arg\max_z f_A(\mathbf{p})\end{bmatrix}))$
- Roof = obj B is mirrored on Z and placed at above surface
** Say obj A is a cave (or "hole object", like tunnel), place obj B (a stalctite for example) on the roof of the tunnel.
The equation becomes $G(f_A(\mathbf{p}), f_B(\mathbf{p} - \begin{bmatrix}0 & 0 & \arg\max_z f_B(\mathbf{p}) + \arg\max_z f_A(\mathbf{p})\end{bmatrix}))$.
- Ground = obj B is placed at lower surface
** Say obj A is a cave (or "hole object", like tunnel), place obj B (a rock for example) on the ground of the tunnel.

The equation becomes $G(f_A(\mathbf{p}), f_B(\mathbf{p} - \begin{bmatrix}0 & 0 & \arg\min_z f_A(\mathbf{p})\end{bmatrix}))$.
- ...

4.5 Returning a material

As we are mixing together multiple primitives with different materials, some may want to collapse the possible materials to a single material. Discretizing the implicit models to a layer-based representation or a material-voxel grid, for example, require a single material for a point in space. No single answer can be found as we will always lose information by doing so. In the backend, the information about each quantities of material may be stored.

4.5.1 Material usage

Storing all the materials that compose a single point in space have many usages. Determining properties such as density, porosity, granularity may be important indicators for future processes. While natural processes like corrosion is applied at the molecular level, keeping the information of the composition of the ground can allow to simulate natural phenomena much easily.

However, when rendering the surface of a 3D model or a terrain, we usually need to reduce the number to textures to blend to one or few. Another use case of the collapse: if we want to transform an implicit model to a material-voxel grid, for example, a unique material may be affected to each voxel.

Defining the final material

If we are in a situation in which we need to reduce the number of materials to a unique material, we have multiple solutions. Each solution has inevitably drawbacks.

- Select the material, or the n materials, with the highest amount. This simple solution requires close to no computation and is intuitive.
- Classify materials in a taxonomy. This allows many new options to implement a voting system.

Usage example: material transformation

As multiple materials may be present at a single position $\mathbf{p}$, we can apply transformation on these materials. We may use the formulation of chemical reactions to describe the phenomena.

An example of transformation can include 1dirt + 1water $\mapsto$ 2mud to describe the mixing of soft materials with liquids may transform this place in a muddy floor. 1water current + 1rock $\mapsto$ 0.5sand simulates the effect of water erosion on solid ground, approximating the desaggregation of rocks into granular materials in addition to corrosion, as the equation is not balanced.

4.6 Graphical representation of environmental objects

As presented in Chapter 3, the environmental objects we used to define our sparse terrain are symbolics and have no geometric representation. It is still possible to translate the

semantic representation into a 3D representation by using a construction tree of the form :

- Root
- Water primitive : 1 large block set at $z = \mathcal{L}$
- "Ground" primitives that are fixed at $z = 0$, aggregated with G_{fixed}
- "Water" primitives aggregated with water primitive, aggregated with G_{roof}
- "Surface" primitives aggregated with G_{stack} to be automatically placed on top of the surface.

While the constuction trees are mainly aimed at combining implicit surfaces, the definition of the primitives can easily be extended to include meshes and implicit volumes. In the rest of this thesis, we will most of the time use implicit volumes for elements that shape the ground and meshes for elements that populate the surface, like vegetation. An exmple of underground 3D feature, karst systems, presented in Chapter D, may require an exception as we will have it a fixed position from the surface. As such, we would apply the $G_{\text{roof}}(f_{\text{surface}}, f_{\text{karst}})$ positionning operator in order to fix the entrance of the karst at the surface.

The implicit volumes used are mainly combinations of tubes, spheres, and boxes, with the inclusion of noise unary nodes to add rugosity at the surface.

CHAPTER 5

Erosion simulation

Figure 5.1: 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.

Abstract

In this chapter, we present a novel particle-based method for simulating erosion on various terrain representations, including height fields, voxel grids, material layers, and implicit terrains. Our approach breaks down erosion into two key processes - terrain alteration and material transport - allowing for flexibility in simulation. We utilize independent particles governed by basic particle physics principles, enabling efficient parallel computation. For increased precision, a vector field can adjust particle speed, adaptable for realistic fluid simulations or user-defined control. We address material alteration in 3D terrains with a set of equations applicable across diverse models, requiring only per-particle specifications for size, density, coefficient of restitution, and sediment capacity. Our modular algorithm is versatile for real-time and offline use, suitable for both 2.5D and 3D terrains.

Contents

Back to summary

5.1 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 (Stachniak and Stuerzlinger, 2005; Smelik, Kraker, et al., 2009; Galin, Guérin, et al., 2019).

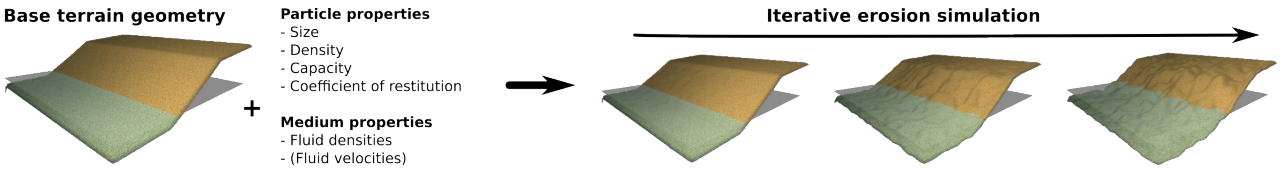


Figure 5.2: Our method require a base geometry, a small number of parameters for the particles and the medium used for the erosion simulation. It can be easily adapted to be compatible with different mediums and terrain representations.

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 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 (Cordonnier, Cani, et al., 2017; Argudo et al., 2020; Cordonnier, Ecormier-nocca, et al., 2018; Cordonnier, Galin, et al., 2017; Cordonnier, Jouvet, et al., 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 5.2). 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.

5.2 Erosion

Erosion is the complex, gradual process through which natural forces such as water, wind, ice, and gravity wear away soil, rock, and sediment, transporting these materials across the landscape and leading to significant changes in topography and landform structure over time. This process, fundamental to both natural and human-modified environments, involves several interacting factors, including the type and composition of surface material, climate conditions (such as precipitation, temperature fluctuations, and wind intensity), topography (including slope steepness and aspect), and the presence of vegetation.

Studying erosion is essential across numerous fields due to its impact on landscapes, ecosystems, infrastructure, and resource sustainability. In agriculture, for example, erosion poses a direct threat to soil fertility and crop productivity, driving the need for conservation practices that retain topsoil and prevent land degradation. In civil and environmental engineering, erosion control measures are fundamental to maintaining the stability of roads, bridges, and coastal structures, safeguarding both urban and rural infrastructure. Similarly, urban planning incorporates erosion management into green infrastructure and stormwater systems to protect densely populated areas from soil loss and water damage. Ecologically, erosion shapes habitat stability and biodiversity, affecting riverine, forest, and coastal ecosystems; as a result, restoration efforts are vital to sustain healthy landscapes and support species diversity. In water resource management, erosion control is integral to preventing sedimentation, preserving water quality, and managing flood risks, which are essential for both environmental health and human safety. Within climate science, erosion is studied for its role in the carbon cycle, land degradation, and desertification, as it influences carbon release and soil loss in vulnerable areas. Erosion is also a key focus in natural disaster risk assessment, where slope stability and riverbank management are crucial for preventing landslides and floods that could endanger communities. Furthermore, in mining and resource extraction,

erosion control mitigates environmental damage by supporting ecosystem recovery and reducing sediment pollution in surrounding areas. [ADD CITATIONS FOR EACH FIELD].

The effects of erosion extend beyond simple landscape alteration. They play a central role in soil formation, nutrient cycling, and the distribution of sediments in various ecosystems.

The main problem with studying erosion is the large amount of factors to take into account in the process. When studying erosion, a range of interconnected properties must be considered to understand its causes, rates, and impacts. Soil and rock characteristics such as texture, structure, cohesion, permeability, and organic matter content influence how easily soil and sediment detach and are transported. Climate factors like rainfall intensity, temperature fluctuations, wind speed, and seasonal changes impact erosion by affecting soil moisture and stability. Topography—including slope gradient, length, elevation, and drainage patterns—affects how water flows and gathers momentum, directly influencing erosion potential. Hydrological factors like surface runoff, infiltration rate, groundwater flow, and wave action are critical in understanding how water drives erosion in riverine and coastal environments.

Biological influences include vegetation cover, root density, and plant types, which stabilize soil, while animal activities like burrowing can increase erosion susceptibility. Human activity, such as agriculture, deforestation, urbanization, and mining, significantly accelerates erosion by disturbing soil and removing natural vegetation. Geological properties like soil depth, rock layering, and prior erosion events, along with chemical characteristics such as mineral composition and soil acidity, also shape erosion patterns, influencing how landscapes respond to environmental forces. Temporal factors—including the duration of exposure, rate of soil formation, seasonal cycles, and the frequency of intense weather events—affect erosion rates over time, leading to varying erosion impacts across regions and ecosystems. These properties collectively provide a comprehensive foundation for understanding and managing erosion processes across diverse landscapes.

Due to the large amount of properties to take into account, each domain uses a subset of the factors, with more or less simplifications in the erosion model. Depending on the use case, the analysis of erosion effects may vary largely. [ADD EXAMPLE OF FIELDS THAT STUDY EROSION THAT HAVE INSTANTANEOUS EFFECTS LIKE LANDSLIDES AND FIELDS THAT USE MILLIONS OF YEARS LIKE TECTONIC ACTIVITY].

In procedural terrain generation, simulating erosion based on soil and rock characteristics (like texture, cohesion, and permeability) allows the algorithm to mimic how different materials break down and redistribute under various conditions. For example, regions with sandy or loosely cohesive soils can be programmed to erode more rapidly, while rocky, cohesive areas remain more stable.

Incorporating climate factors (rainfall, wind, and temperature) helps to model how landscapes evolve in different biomes—such as dry, windy deserts versus lush, wet valleys. Simulating topographic influences like slope gradient and drainage patterns enables the formation of natural-looking valleys, cliffs, and river networks, while accounting for hydrological factors like runoff and groundwater flow creates realistic water paths and sediment deposits.

Procedural algorithms can also use biological factors to define erosion resistance based on vegetation cover, with densely vegetated areas exhibiting slower erosion rates and barren regions eroding faster. Including human activity effects, such as altered runoff or deforestation, allows for terrain that mirrors landscapes influenced by development or agriculture, adding further realism. Finally, using temporal properties to simulate seasonal erosion and landscape changes enables the terrain to evolve naturally over time, creating dynamic environments where erosion patterns shift as weather and vegetation change.

5.3 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.

5.3.1 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 (Galin, Guérin, et al., 2019).

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.

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. (de Araújo et al., 2015)

Layered models

Layered models are a type of volumetric representation that encode different material layers within the terrain and are defined by a function $\mu : \mathbb{R}^3 \mapsto \mathcal{M}$, where $\mathcal{M}$ 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) (Laine and Karras, 2010) or voxel Directed Acyclic Graphs (DAG) (Villanueva et al.,

2017; Careil et al., 2020). 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 $\mu : \mathbb{R}^3 \mapsto \mathcal{M}$, 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.

5.3.2 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 (Musgrave et al., 1989; Beneš and Forsbach, 2001; 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, Těšínský, 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.

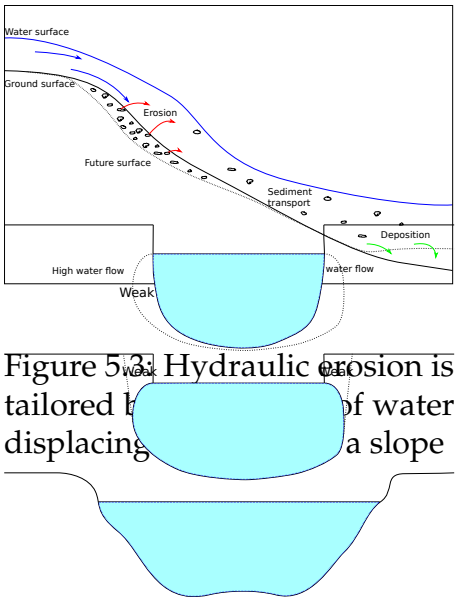


Figure 5.3: Hydraulic erosion is tailored b f water displacing a slope

Rivers

Rain

Corrosion

Figure 5.4: .

Landslides

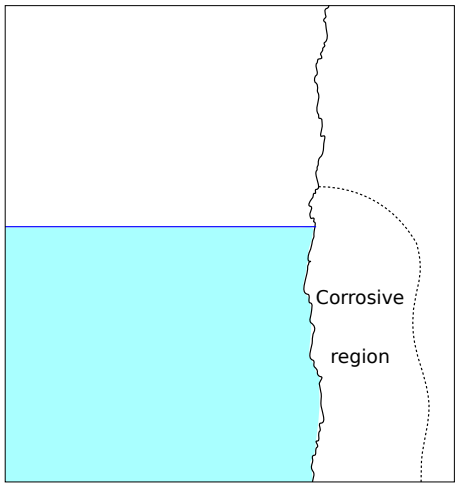


Figure 5.5: .

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 (Roa and Benes, 2004; Paris, Peytavie, et al., 2019) 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 (Cordonnier, Braun, et al., 2016; Cordonnier, Cani, et al., 2017; Cordonnier, Ecormier-nocca, et al., 2018; Cordonnier, Galin, et al., 2017; Argudo et al., 2020). 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. *Glaciers*

Animals

Storms

Volcanos

5.3.3 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.

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f} \tag{5.1}$$

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:

$$\nabla \cdot \mathbf{u} = 0 \tag{5.2}$$

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 (Stam, 1999). 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 (Brackbill et al., 1988). 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) (Müller et al., 2003) 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 (Koschier et al., 2022). 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.

We can find improvements (Roose et al., 2011), can represent may elements (Iwasaki et al., 2010), either fluid or sediments (Lenaerts and Dutré, 2009), so it is often used to simulate accurately water bodies (Nikeghbali and Omidvar, 2018).

Some other models exist, and are often proposed in OpenFOAM, either using grids, or particles, or an hybrid version (Caretto et al., 1973) [ADD OTHER REFERENCES]. But some works tend in directions that are more "procedural" like the use of cellular automata (Boldea, 2009; Cattaneo and Jocher, 2005), while other toward a more modern approach with the use of deep learning (Tompson et al., 2017) [ADD OTHER REFERENCES].

5.4 Particle erosion

Erosion occurs in three stages: material detachment, transport and deposition (respectively in red, black and green in Figure 5.6). 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.

5.4.1 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

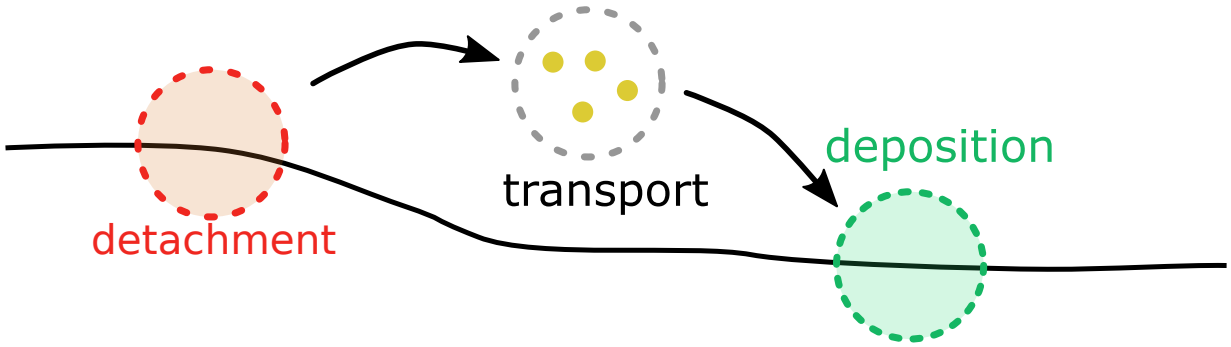
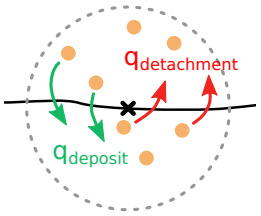


Figure 5.6: Three steps of the erosion process from the sediment point of view: detachment from its original location - dotted red circle -, transport in a fluid - dotted black circle -, deposition at a new location - dotted green circle.

density to be 1000 kg m^{-3} and that of air to be 1 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.

5.4.2 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:

$$\tau = K\theta^n \tag{5.3}$$

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 (5.3) by

$$\varepsilon = K_\varepsilon(\tau - \tau_{\text{critical}})^a \tag{5.4}$$

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 5.9). We then use Equation (5.4):

$$q_{\text{detachment}} = \varepsilon\frac{2\pi R^3}{3} \tag{5.5}$$

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 (5.9).

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 (5.9). 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 $\omega \in [0, 1]$:

$$q_{\text{deposit}} = \omega||w_s||. \tag{5.6}$$

5.4.3 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:

$$\vec{F}_{\text{ext}} = V\vec{g}(\rho_{\text{particle}} - \rho_{\text{fluid}}). \tag{5.7}$$

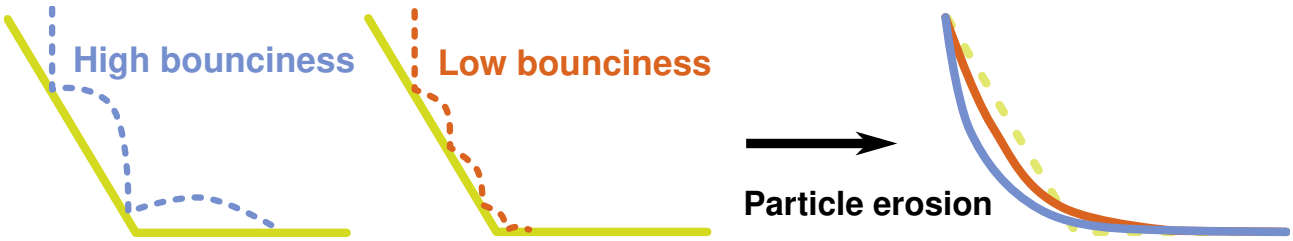


Figure 5.7: The coefficient of restitution affects the amount of energy absorbed from the particle when hitting the ground. Here, rain is applied on an initial slope (yellow). Only two particles are displayed, with a high (blue) and low (red) coefficient of restitution. The resulting slope after erosion is displayed in blue and red (right).

The particle velocity v can be integrated from Equation (5.7) by:

$$v = \int \vec{F}_{\text{ext}}\, dt + w_s + v_0, \tag{5.8}$$

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

$$w_s = \frac{2}{9}\vec{g}R^2\frac{(\rho_{\text{particle}} - \rho_{\text{fluid}})}{\mu} f(C). \tag{5.9}$$

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 5.7 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 5.7 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 fluid simulation, a uniform vector field, or an artistic motion field. We modify Equation Equation (5.8) such that the particle's speed will be influenced by the velocity field as follows:

$$v = \int \vec{F}_{\text{ext}}\, dt + w_s + \alpha v_{\text{fluid}} + v_0, \tag{5.10}$$

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 5.11, allows us to find a balance between achieving realistic flow simulations and maintaining computational efficiency.

5.5 Our erosion method

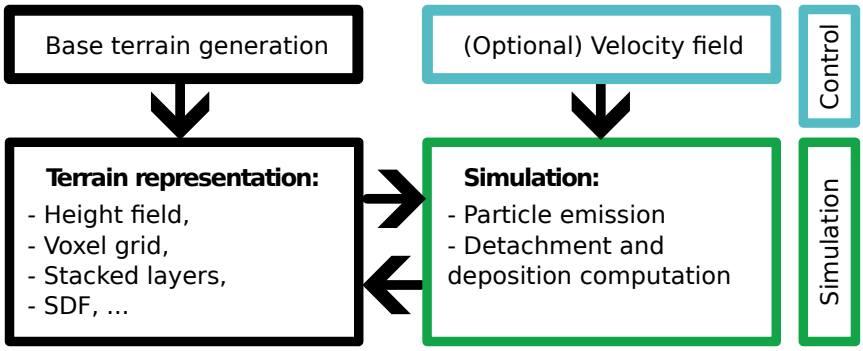


Figure 5.8: Our overall pipeline: our erosion process compute matter displacement of a terrain using an arbitrary representation as long as intersections between particles and the ground can be detected. An optional velocity field, provided by the user, guides the particles trajectories. We propose surface 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 5.8). 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

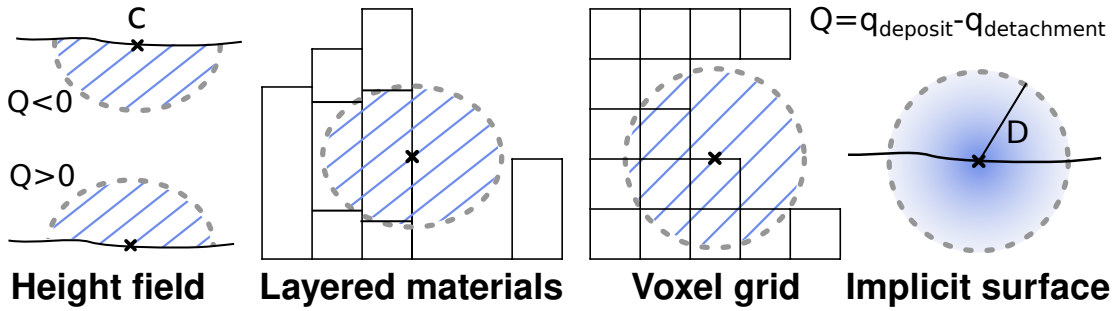


Figure 5.9: Illustration of the material detachment in the (half-)sphere at contact point C (cross) on different representations. (height field) When $Q < 0$ material detachment happen in the bottom scaled half sphere of the particle's contact with the ground, while the deposition is applied on the upper 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.

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).

For all the representations, the amount of material absorbed by the particle, i.e. the erosion value $q_{\text{detachment}}$ from Equation (5.5), 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 (5.6) 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 5.9). In the following sections, we will describe the strategies used to modify the amount of matter for different representations.

5.5.1 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

$$\Delta h(\mathbf{p}) = \frac{\sqrt{R^2 - d^2}}{\alpha} = \frac{Q}{\frac{2}{3}\pi R^3}\sqrt{R^2 - d^2} \tag{5.11}$$

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

5.5.2 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 (5.11). At a distance d from the particle, the height is defined as:

$$H(d) = \frac{|Q|}{\frac{2}{3}\pi R^3}\sqrt{R^2 - d^2}. \tag{5.12}$$

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.

5.5.3 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:

$$\Delta f(\mathbf{p}) = \frac{3Q}{\pi}\frac{(1-d)}{R} \tag{5.13}$$

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 Chapter B).

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 (5.13) 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.

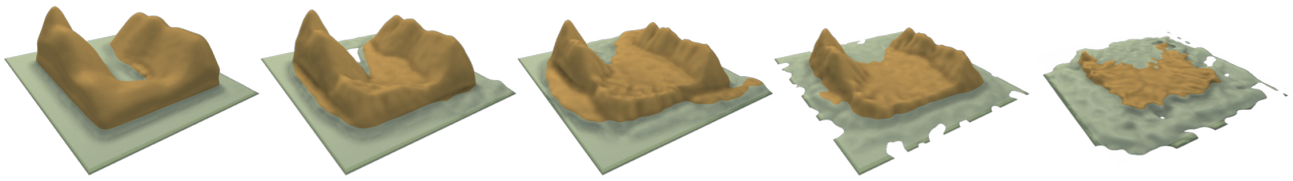


Figure 5.10: Our erosion method is applied iteratively on a completely synthetic island, the terrain is altered to obtain a plausible shape by forming rills. The use of particles with hydraulic densities dropped from the sky results in a strong erosion on the sides of the mountains, and the particles 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.

Other implicit terrains

are present in the literature, notably a 2.5D representation based on the surface gradient (Guérin, Peytavie, 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 (5.11) that gives:

$$\nabla h' = -\frac{Q}{\frac{2}{3}R^3}\frac{1}{\sqrt{R^2-d^2}}\vec{CP} \tag{5.14}$$

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:

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

5.5.4 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 (5.13), 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:

$$\Delta f(\mathbf{p}) = Q\frac{(1-\frac{d}{R})}{\sum(1-\frac{d}{R})}. \tag{5.15}$$

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:

$$\Delta hits = \lfloor \alpha \Delta f \rfloor \tag{5.16}$$

with Δf the erosion per voxel computed using Equation (5.15) 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 (5.11), can be used to erode 2D scalar field-based representations. Similarly, our proposition for SDF Equation (5.13) enables erosion for continuous 3D scalar fields and voxels Equation (5.15) for discrete 3D scalar fields respectively.

5.6 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 5.17, parameters used are available at Table 5.1. It is important to note that all erosion examples presented in this section are available for any 3D terrain representation. 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.

5.6.1 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

Name	Rep.	Dimensions	Res	$\#P$	$\#N$	R	COR	ρ_{particle}	C_{factor}	ε	ω	Vel field	t
Rain	H	100x100	20	100	10	1.0	1.0	1000	10.0	2.5	0.3	None	4.0
Coastal	DV	100x100x30	10	80	3	5	0.1	500	10.0	5.0	0.5	Uniform	0.5
Meanders	I	N/A	N/A	10	20	5.0	1.0	1000	1.0	1.0	1.0	(1)	1
River	H	100x100	5	100	50	1.5-5	0.5	900	0.1	1.0	1.0	None	2.5
Landslide	H	100x100	20	200	10	2.5	0.2	500	0.1	1.0	1.0	None	4
Volcano	DV	100x100x40	50	150	30	1.0	5.0	2000	1.0	1.0	5.0	None	0.8
Karst	BV	100x100x50	2	1000	40	5	0.5	500	10.0	5.0	0.5	Uniform	20
Tunnel	DV	100x100x50	1	100	100	2.5	0.1	500	1.0	1.0	1.0	None	0.8
Wind	DV	100x100x50	0.2	100	10	1.5	0.9	1.5	1.0	1.0	1.0	(Paris, Galin, et al., 2019)	0.5
Underwater	H	100x100	10	100	50	2.5	0.9	1000	1.0	1.0	1.0	(Stam, 2003a)	4

Table 5.1: Parameters used for the generation of the terrains presented in Figure 5.17, 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 $v_{\text{fluid}}(\mathbf{p}) = [0\ sin(\mathbf{p}_x)\ 0]^T$.

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 5.17: Rain* with a computation time of 4 seconds.
Using this erosion parameters in combination with water bodies results in different outcomes (Figure 5.10). 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.

5.6.2 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 5.17: 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 5.13. A shaded version of this effect is presented in Figure 5.1.

5.6.3 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 5.17: Meanders*).
Figure 5.17: 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.

5.6.4 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 5.17: 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 5.17: Volcano*.

5.6.5 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, Guérin, 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 5.17: 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 5.17: 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.

5.6.6 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, Peytavie, et al., 2019) 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 5.17: 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, Peytavie, et al., 2019) is used on light particles, the simulation is fast thanks to the low number of collisions each particle has with the ground.

5.6.7 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 5.17: 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 (5.10) (Figure 5.11).

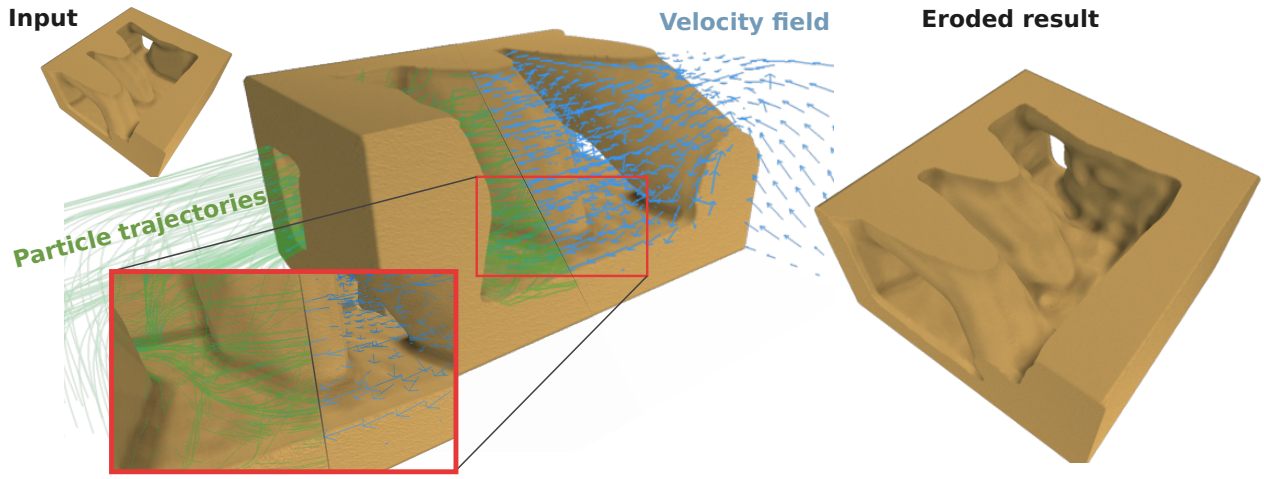


Figure 5.11: A complex water flow simulation is computed using OpenFOAM. Particle trajectories (green) are highly affected by the fluid velocity (blue). Most the terrain exposed surfaces is eroded (bottom).

5.6.8 Multiple phenomena

A terrain eroded with multiple erosion phenomena applied on a 500x500x50 density-voxel grid is illustrated in Figure 5.12. Here, water-density particles are 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.

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

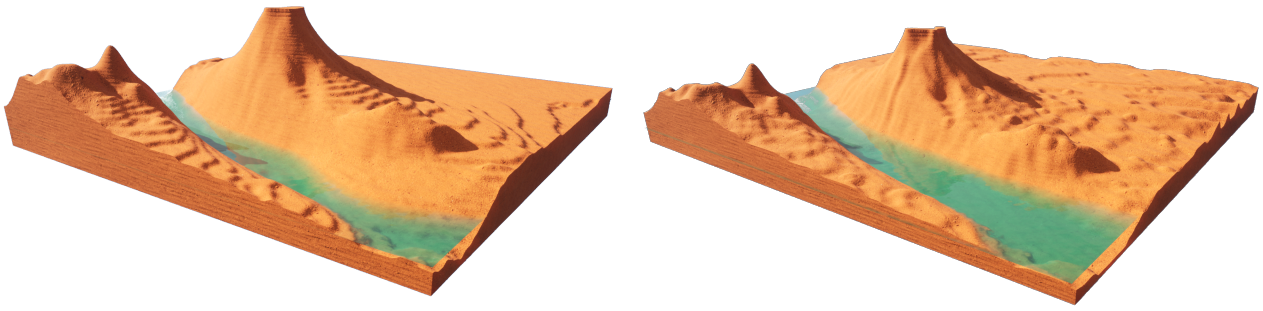


Figure 5.12: Multiple erosion types can be combined. On an initial synthetic 500x500x50 density voxel grid, the a wind erosion is applied on the surface of the terrain while hydraulic erosion shapes the rills and the base of the mountains. A water current digs its borders and spreads sediments at the bottom.

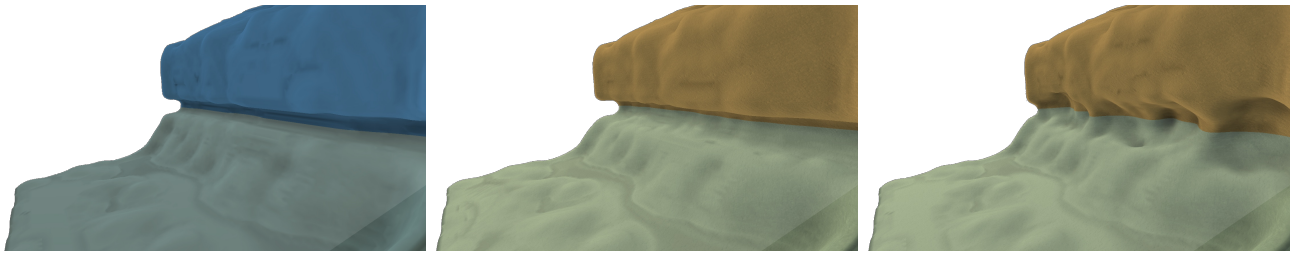


Figure 5.13: The algorithm proposed by (Paris, Galin, et al., 2019) allows for the simulation of coastal erosion (left) that we can reproduce almost identically by allowing our particles to collide only once with the ground and applying only erosion (center). If we apply our erosion with the full tracking of our particles and using deposition, we can achieve more diverse results (right).

5.7.1 Coastal erosion on implicit terrain representation

Paris, Galin, 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 5.13);

5.7.2 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 5.14).

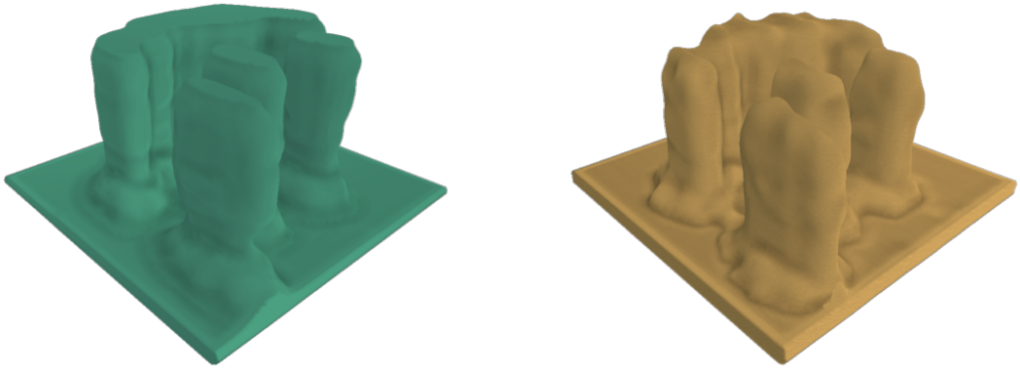


Figure 5.14: The algorithm proposed by (Beardall et al., 2010) allows for an efficient simulation of the spheroidal erosion, making the creation of gobelins on voxel grids in a plausible way (left). Our algorithm naturally erodes the most exposed areas of the terrain when particles are affected by the wind (right).

5.7.3 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 5.15).

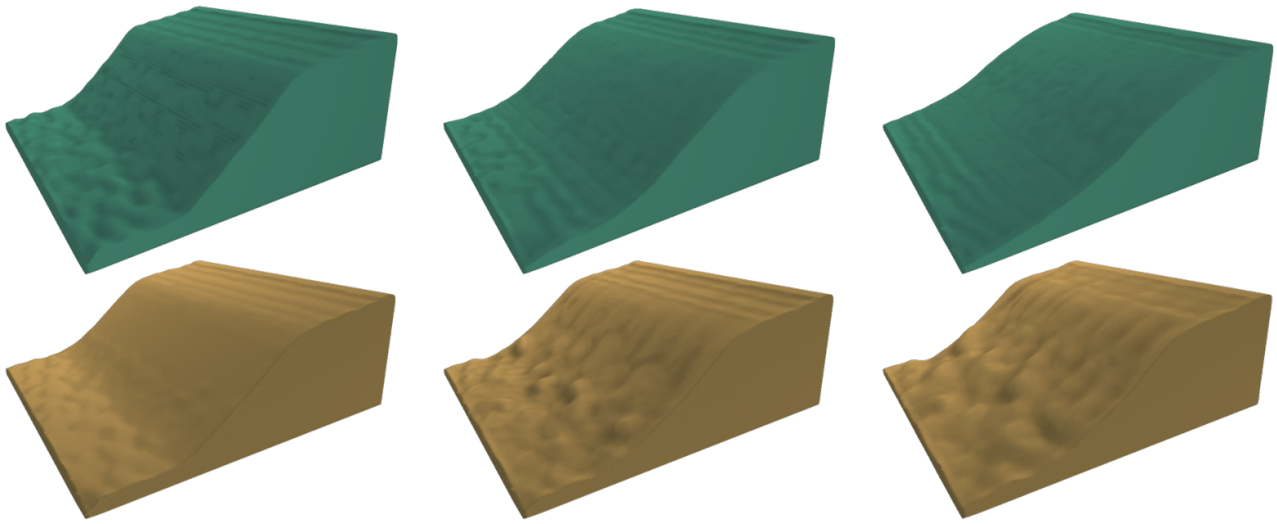


Figure 5.15: While our resulting geometry on the hydraulic erosion (bottom) is less smoothed than the one proposed proposed by (Mei et al., 2007) (top), our method allows the application on more terrain representations than the height fields only.

5.7.4 Wind erosion on stacked materials representation

Paris, Peytavie, 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, Cani, et al., 2017) that are not affected by abrasion(Paris, Peytavie, et al., 2019). 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 5.16.

5.8 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.

5.8.1 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

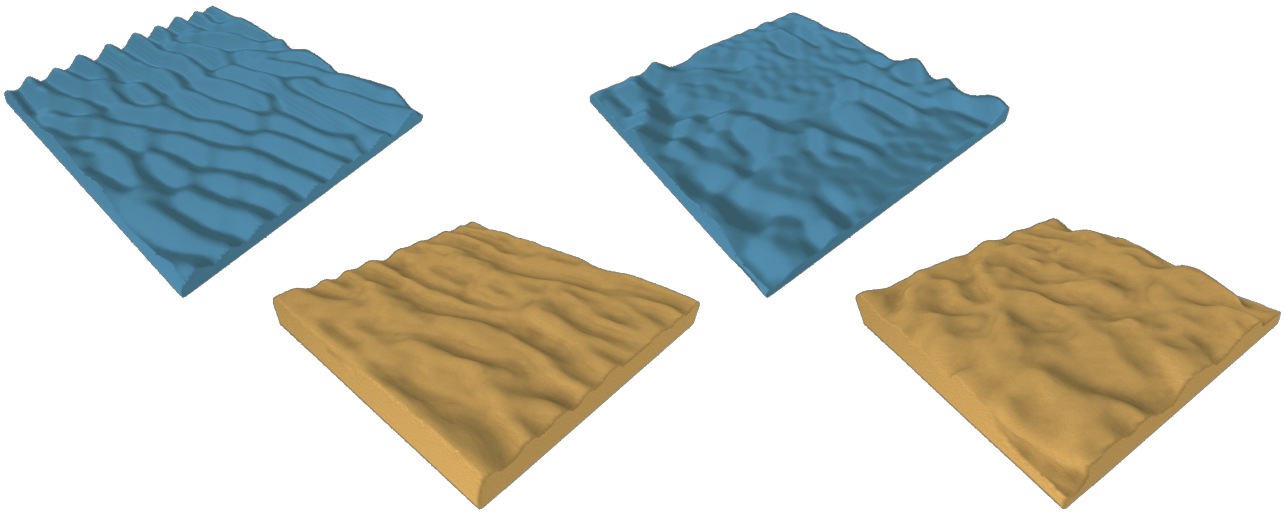


Figure 5.16: The algorithm from Paris 2020 allow the generation of desertscapes (top), which we can (at least partially) reproduce with our erosion simulation (bottom). The different effects are achieved by affecting the wind direction and strength.

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.

5.8.2 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.

5.8.3 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.

5.9 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.

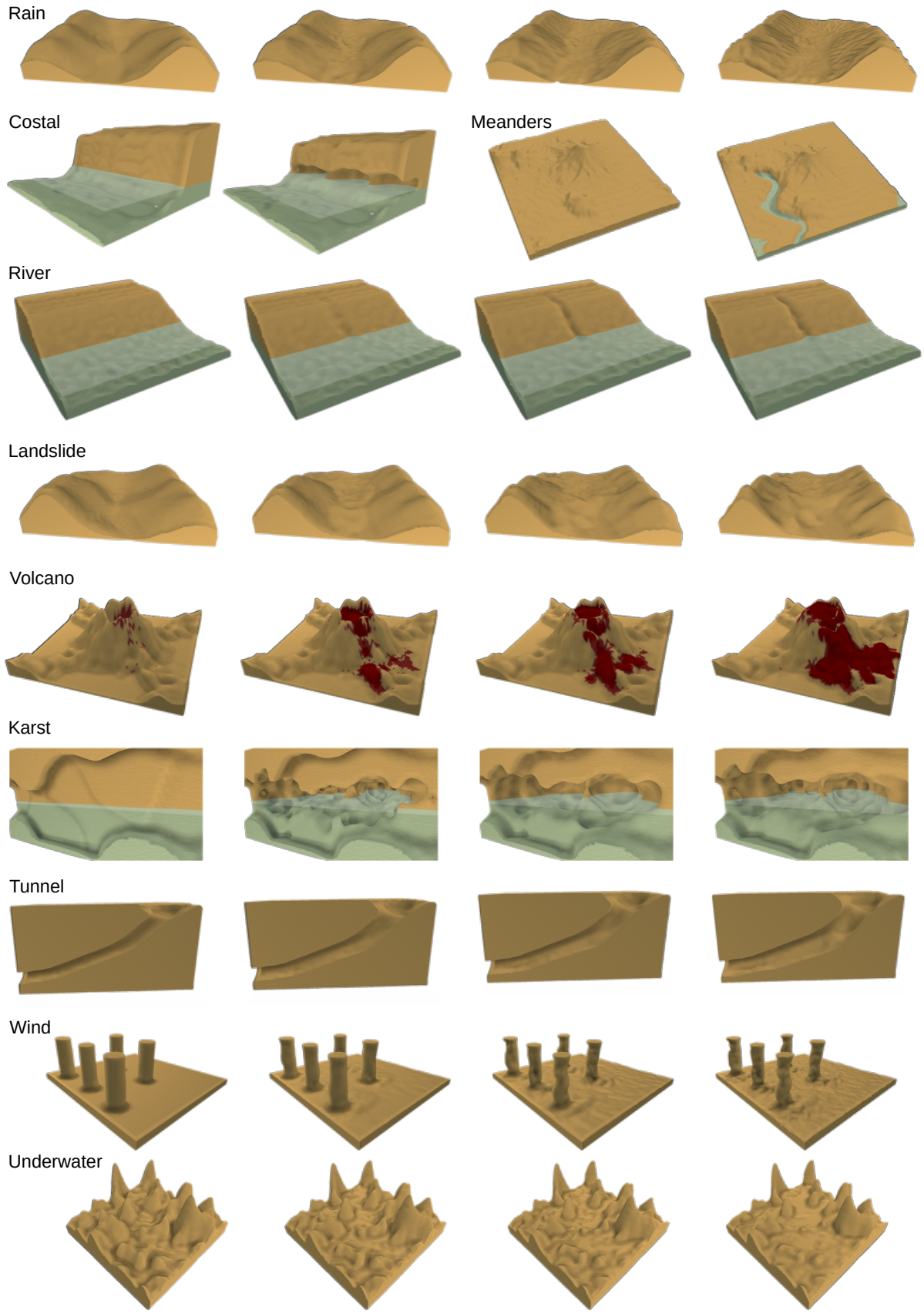


Figure 5.17: Erosion processes results on various representations presented in section Section 5.6. Used parameters used are detailed in Table 5.1.

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. 54).

Amawy, E. A., & 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. 52).

Angles, B., Tarini, M., Wyvill, B., Barthe, L., & Tagliasacchi, A. (2017). Sketch-based implicit blending. *ACM Transactions on Graphics*, *36*, 1–13. https://doi.org/10.1145/3130800.3130825 (cit. on p. 74).

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. 82, 89).

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. 97).

Bari, E., Nipa, N. J., & Roy, B. (2021). Association of vegetation indices with atmospheric & biological factors using modis time series products. *Environmental Challenges*, *5*. https://doi.org/10.1016/j.envc.2021.100376 (cit. on p. 132).

Barthe, L., Wyvill, B., & Groot, E. D. (2004). Controllable binary csg operators for "soft objects". *International Journal of Shape Modeling*, *10*, 135–154. https://doi.org/10.1142/S021865430400064X (cit. on p. 74).

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. 100, 106).

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. 88, 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 pp. 32, 88).

Bernhardt, A., Barthe, L., Cani, M.-P., & Wyvill, B. (2010). Implicit blending revisited. *Computer Graphics Forum*, *29*, 367–375. https://doi.org/10.1111/j.1467-8659.2009.01606.x (cit. on p. 74).

Boldea, C.-R. (2009). A particle cellular automata model for fluid simulations. *Math. Comp. Sci. Ser*, *36*, 35–41 (cit. on pp. 11, 17, 92).

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. 54).

Braben, D., & Bell, I. (1984). Elite. (Cit. on p. 5).

Brackbill, J. U., Kothe, D. B., & Ruppel, H. M. (1988). Flip: A low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications*, *48*, 25–38. https://doi.org/10.1016/0010-4655(88)90020-3 (cit. on pp. 16, 91).

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. 53).

Careil, V., Billeter, M., & Eisemann, E. (2020). Interactively modifying compressed sparse voxel representations. *Computer Graphics Forum*, *39*, 111–119. https://doi.org/10.1111/cgf.13916 (cit. on pp. 10, 88).

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 pp. 17, 92, 104).

Cattaneo, G., & Jocher, U. (2005). Cellular automata for 2d and 3d fluid-dynamics simulations. (Cit. on pp. 11, 17, 92).

Chan, T. F., & Vese, L. A. (2001). Active contours without edges. *IEEE Transactions on image processing*, *10* (cit. on p. 62).

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 pp. 132, 133).

Collon, P., Bernasconi, D., Vuilleumier, C., & Renard, P. (2021). Corrigendum to "statistical metrics for the characterization of karst network geometry and topology" [geomorphology (2017) 283: 122–142]. *Geomorphology*, *389*, 107848. https://doi.org/10.1016/j.geomorph.2021.107848 (cit. on p. 132).

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. 133).

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. 33, 89).

Cordonnier, G., Cani, M.-P., Beneš, B., Braun, J., & Galin, É. (2017). 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. 33, 82, 89, 107).

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. 82, 89).

Cordonnier, G., Galin, É., Gain, J., Beneš, B., Guérin, É., Peytavie, A., & Cani, M.-P. (2017). Authoring landscapes by combining ecosystem and terrain erosion simulation. *ACM Transactions on Graphics*, *36*. https://doi.org/10.1145/3072959.3073667 (cit. on pp. 33, 82, 89).

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/10.1145/3592422Ãŕ (cit. on pp. 11, 82).

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. 11).

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. 67).

Daly, R. A. . (1915). The glacial-control theory of coral reefs. *Proceedings of the American Academy of Arts and Sciences*, *51*, 157–251 (cit. on p. 28).

Dam, P., Duarte, F., & Raposo, A. (2019). Terrain generation based on real world locations for military training and simulation. *Brazilian Symposium on Games and Digital Entertainment, SBGAMES*, *2019-Octob*, 173–181. https://doi.org/10.1109/SBGames.2019.00031 (cit. on p. 6).

Darwin, C., & Jackson, C. (1842). The structure and distribution of coral reefs: Being the first part of the geology of the voyage of the 'beagle,' under the command of capt. fitzroy, r. n., during the years 1832 to 1836. https://doi.org/10.2307/1797986 (cit. on pp. 23, 27).

de Araújo, B. R., Lopes, D. S., Jepp, P., Jorge, J. A., & Wyvill, B. (2015). A survey on implicit surface polygonization. *ACM Computing Surveys*, *47*, 1–39. https://doi.org/10.1145/2732197 (cit. on pp. 9, 87).

Design, A. (1980). Rogue. (Cit. on p. 5).

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

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. 67).

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. 28).

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 pp. 6, 58).

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

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

Fedkiw, R., & Jensen, H. W. (2001). Visual simulation of smoke. *SIGGRAPH 2001 Conference Proceedings, Annual Conference Series*, 15–22 (cit. on p. 3).

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. 133).

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. 33, 34, 53).

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. 8, 53, 81, 85).

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. 133).

Games, B. 1. (2006). Dwarf fortress. (Cit. on p. 5).

Génevaux, J.-D., Galin, É., Peytavie, A., Guérin, É., Briquet, C., Grosbellet, F., & Beneš, B. (2015). Terrain modelling from feature primitives. https://doi.org/https://doi.org/10.1111/cgf.12530 (cit. on p. 74).

Getreuer, P. (2012). Chan-vese segmentation. *Image Processing On Line*, *2*, 214–224. https://doi.org/10.5201/ipol.2012.g-cv (cit. on p. 62).

Gobron, S., & Chiba, N. (2001). Crack pattern simulation based on 3d surface cellular automata. *Visual Computer*, *17*, 287–309. https://doi.org/10.1007/s003710100099 (cit. on pp. 4, 6).

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. 58, 65).

Gouy, A., Collon, P., Bailly-Comte, V., Galin, E., Antoine, C., Thebault, B., & Landrein, P. (2024). Karstnsim: A graph-based method for 3d geologically-driven simulation of karst networks. *Journal of Hydrology*, *632*. https://doi.org/10.1016/j.jhydrol.2024.130878 (cit. on p. 133).

Greene, N. (1989). Voxel space automata: Modeling with stochastic growth processes in voxel space. (Cit. on p. 11).

Groot, E. D., Wyvill, B., Barthe, L., Nasri, A., & Lalonde, P. (2014). Implicit decals: Interactive editing of repetitive patterns on surfaces. *Computer Graphics Forum*, *33*, 141–151. https://doi.org/10.1111/cgf.12260 (cit. on p. 74).

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. 54).

Guérin, E., Galin, E., Grosbellet, F., Peytavie, A., & Génevaux, J.-D. (2016). Efficient modeling of entangled details for natural scenes. *Computer Graphics Forum*, *35*, 257–267. https://doi.org/10.1111/cgf.13023 (cit. on p. 74).

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 p. 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. 49, 54).

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 pp. 35, 53).

Hädrich, T., Banuti, D. T., Pałubicki, W., Pirk, S., & Michels, D. L. (2021). Fire in paradise. *ACM Transactions on Graphics*, *40*. https://doi.org/10.1145/3450626.3459954 (cit. on pp. 3, 6).

Hawick, K. A. (2014). Modelling flood incursion and coastal erosion using cellular automata simulations. *Proceedings of the IASTED International Conference on Environmental Management and Engineering, EME 2014*, 158–165. https://doi.org/10.2316/P.2014.821-005 (cit. on p. 32).

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. 108).

Huang, J., Ge, Z., Huang, Y., Tang, X., Shi, Z., Lai, P., Song, Z., Hao, B., Yang, H., & Ma, M. (2022). Climate change and ecological engineering jointly induced vegetation greening in global karst regions from 2001 to 2020. *Plant and Soil*, *475*, 193–212. https://doi.org/10.1007/s11104-021-05054-0 (cit. on p. 131).

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. 52).

Hudák, M., & Ďurikovič, R. (2011). Terrain models for mass movement erosion. *Theory and Practice of Computer Graphics 2011, TPCG 2011 - Eurographics UK Chapter Proceedings*, 9–16. https://doi.org/10.2312/LocalChapterEvents/TPCG/TPCG11/009-016 (cit. on p. 6).

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

Iwasaki, K., Uchida, H., Dobashi, Y., & Nishita, T. (2010). Fast particle-based visual simulation of ice melting. *Computer Graphics Forum*, *29*, 2215–2223. https://doi.org/10.1111/j.1467-8659.2010.01810.x (cit. on pp. 4, 16, 92).

Jaquet, O., Siegel, P., Klubertanz, G., & Benabderrhamane, H. (2004). Stochastic discrete model of karstic networks. *Advances in Water Resources*, *27*, 751–760. https://doi.org/10.1016/j.advwatres.2004.03.007 (cit. on p. 133).

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).

Jouves, J., Viseur, S., Arfib, B., Baudement, C., Camus, H., Collon, P., & Guglielmi, Y. (2017). Speleogenesis, geometry, and topology of caves: A quantitative study of 3d karst conduits. *Geomorphology*, *298*, 86–106. https://doi.org/10.1016/j.geomorph.2017.09.019 (cit. on p. 133).

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. 53).

Kass, M., Witkin, A., & Terzopoulos, D. (1988). Snakes: Active contour models. *International Journal of Computer Vision*, *1*, 321–331. https://doi.org/10.1007/BF00133570 (cit. on pp. 60, 61).

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

Ketabchi, K., Runions, A., & Samavati, F. F. (2016). 3d maquetter: Sketch-based 3d content modeling for digital earth. *Proceedings - 2015 International Conference on Cyberworlds, CW 2015*, 98–106. https://doi.org/10.1109/CW.2015.41 (cit. on p. 54).

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 pp. 16, 92, 96).

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. 82, 88, 93, 96, 102).

Laine, S., & Karras, T. (2010). Efficient sparse voxel octrees. *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 55–63. https://doi.org/10.1145/1730804.1730814 (cit. on pp. 10, 87).

Lejeune, C. (2021). Génération et analyse de tests pour les systèmes autonomes, 170 (cit. on p. 6).

Lenaerts, T., & Dutré, P. (2009). Mixing fluids and granular materials. *Computer Graphics Forum*, *28*, 213–218. https://doi.org/10.1111/j.1467-8659.2009.01360.x (cit. on pp. 16, 92).

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

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. 54).

Lima, F. T., Brown, N. C., & Duarte, J. P. (2022). A grammar-based optimization approach for walkable urban fabrics considering pedestrian accessibility and infrastructure cost. *Environment and Planning B: Urban Analytics and City Science*, *49*, 1489–1506. https://doi.org/10.1177/23998083211048496 (cit. on p. 4).

Louis, S., Andreu, D., Godary-dejean, K., Lapierre, L., Louis, S., Andreu, D., Godary-dejean, K., Lapierre, L., Simulator, H. I. L., Ouis, S. L., Ndreu, D. A., Ejean, K. G. O., & Apierre, L. L. (2019). Hil simulator for auv with contract. *CAR: Control Architectures of Robots* (cit. on p. 6).

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. 54).

Maxis. (2008). Spore. (Cit. on p. 5).

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. 88, 106, 107).

Menshutina, N. V., Kolnoochenko, A. V., & Lebedev, E. A. (2020). Cellular automata in chemistry and chemical engineering. https://doi.org/10.1146/annurev-chembioeng (cit. on p. 11).

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. 11).

Michel, É., & Boubekeur, T. (2023). Mesogen: Designing procedural on-surface stranded mesostructures. *Proceedings - SIGGRAPH 2023 Conference Papers*. https://doi.org/10.1145/3588432.3591496 (cit. on p. 4).

Müller, M., Charypar, D., & Gross, M. (2003). Particle-based fluid simulation for interactive applications. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA 2003* (cit. on pp. 16, 92).

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. 53, 81, 88).

Narain, R. (2011). Visual modeling and visualisation of multiscale phenomena (cit. on p. 3).

Neidhold, B., Wacker, M., & Deussen, O. (2005). Interactive physically based fluid and erosion simulation. *Natural Phenomena*, 25–32 (cit. on pp. 82, 88).

Nikeghbali, P., & Omidvar, P. (2018). Application of the sph method to breaking and undular tidal bores on a movable bed. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, *144*. https://doi.org/10.1061/(asce)ww.1943-5460.0000424 (cit. on pp. 17, 92).

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. 106).

Olsen, J. (2004). Realtime procedural terrain generation. *Department of Mathematics And Computer Science*, 20 (cit. on p. 81).

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. 107).

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. 67).

Pardo-Igúzquiza, E., Dowd, P. A., Xu, C., & Durán-Valsero, J. J. (2012). Stochastic simulation of karst conduit networks. *Advances in Water Resources*, *35*, 141–150. https://doi.org/10.1016/j.advwatres.2011.09.014 (cit. on p. 133).

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. 103, 133, 134).

Paris, A., Peytavie, A., Guérin, E., Argudo, O., & Galin, E. (2019). Desertscape simulation. *Computer Graphics Forum*, *38*, 47–55. https://doi.org/10.1111/cgf.13815 (cit. on pp. 64, 89, 103, 107).

Paris, A. (2023). Modeling and simulating virtual terrains (cit. on p. 18).

Paris, A., Galin, E., Peytavie, A., Guérin, E., & Gain, J. (2019). Terrain amplification with implicit 3d features. *ACM Transactions on Graphics*, *38*, 1–15. https://doi.org/10.1145/3342765 (cit. on pp. 11, 101, 105).

Paris, A., Galin, É., Peytavie, A., Guérin, É., & Gain, J. (2021). Amplification de terrains avec des caractéristiques implicites 3d (cit. on p. 74).

Park, M. J. (2010). Guiding flows for controlling crowds. *Visual Computer*, *26*, 1383–1391. https://doi.org/10.1007/s00371-009-0415-4 (cit. on p. 3).

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. 11, 33).

Perlin, K. (1985). An image synthesizer. *ACM SIGGRAPH Computer Graphics*, *19*, 287–296. https://doi.org/10.1145/325165.325247 (cit. on p. 30).

Perlin, K. (2001). Noise hardware. *Real-Time Shading SIGGRAPH Course Notes* (cit. on p. 30).

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. 88, 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. 52).

Pytel, A., & Mann, S. (2015). Procedural modeling of cave-like channels. (Cit. on p. 133).

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. 109).

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. 95).

Rigaudière, D., Gesquière, G., & Faudot, D. (2000). Shape modelling with skeleton based implicit primitives. *Methods* (cit. on p. 108).

Roa, T., & Benes, B. (2004). Simulating desert scenery. *Winter School of Computer Graphics SHORT communication Papers Proceedings*, 17–22 (cit. on pp. 89, 107).

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 pp. 16, 92, 96).

Roudier, P., Peroche, B., & Perrin, M. (1993). Landscapes synthesis achieved through erosion and deposition process simulation. *Computer Graphics Forum*, *12*, 375–383. https://doi.org/10.1111/1467-8659.1230375 (cit. on p. 32).

Roudier, P. (1993). Synthèse de paysages réalistes par simulation de processus d'érosion (cit. on p. 81).

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. 133).

Schmidt, R., Wyvill, B., Sousa, M. C., & Jorge, J. A. (2006). Shapeshop: Sketch-based solid modeling with blobtrees. *SIGGRAPH 2006 - ACM SIGGRAPH 2006 Courses*, 1–10. https://doi.org/10.1145/1185657.1185775 (cit. on p. 74).

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 pp. 11, 32, 33).

Schroeder, W. J., Lorensen, W. E., & Linthicum, S. (1994). Implicit modeling of swept surfaces and volumes. *Proceedings Visualization*, 40–45. https://doi.org/10.1109/visual.1994.346339 (cit. on p. 76).

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. 52).

Shi, Z., Fonseca, J. A., & Schlueter, A. (2017). A review of simulation-based urban form generation and optimization for energy-driven urban design. *Building and Environment*, *121*, 119–129. https://doi.org/10.1016/j.buildenv.2017.05.006 (cit. on p. 4).

Sims, K. (1990). Particle animation and rendering using data parallel computation. *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, *24*, 405–413. https://doi.org/10.1145/97879.97923 (cit. on p. 54).

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. 81).

Smelik, R. M., Tutenel, T., Bidarra, R., & Benes, B. (2014). A survey on procedural modelling for virtual worlds. *Computer Graphics Forum*, *33*, 31–50. https://doi.org/10.1111/cgf.12276 (cit. on p. 54).

Smelik, R. M., Tutenel, T., Kraker, K. J. D., & Bidarra, R. (2010). Interactive creation of virtual worlds using procedural sketching. *Eurographics* (cit. on p. 54).

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. 81).

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 pp. 15, 91, 104).

Stam, J. (2003a). Flows on surfaces of arbitrary topology. *ACM Transactions on Graphics*, *22*, 724–731. https://doi.org/10.1145/882262.882338 (cit. on p. 101).

Stam, J. (2003b). Real-time fluid dynamics for games. *Proceedings of the Game Developer Conference*, *18*, 17 (cit. on p. 3).

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. 95).

Studios", " (2011). Minecraft. (Cit. on p. 6).

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. 97).

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 pp. 34, 53).

Tasse, F. P., Emilien, A., Cani, M.-P., Hahmann, S., & Bernhardt, A. (2014). First person sketch-based terrain editing. (Cit. on p. 33).

Tasse, F. P., Emilien, A., Cani, M.-P., Hahmann, S., & Dodgson, N. (2014). Feature-based terrain editing from complex sketches. *Computers & Graphics*, *45*, 101–115. https://doi.org/10.1016/j.cag.2014.09.001 (cit. on p. 34).

Temuçin, M. B., Kocabaş, İ., & Oğuz, K. (2020). Using cellular automata as a basis for procedural generation of organic cities. *European Journal of Engineering Research and Science*, *5*, 116–120. https://doi.org/10.24018/ejers.2020.5.12.2293 (cit. on p. 5).

Tompson, J., Schlachter, K., Sprechmann, P., & Perlin, K. (2017). Accelerating eulerian fluid simulation with convolutional networks. *34th International Conference on Machine Learning, ICML 2017*, *7*, 5258–5267 (cit. on pp. 17, 92).

Turk, G., Dinh, H. Q., O'Brien, J., & Yngve, G. (2001). Implicit surfaces that interpolate. *Proceedings International Conference on Shape Modeling and Applications*, 62–71. https://doi.org/10.1109/SMA.2001.923376 (cit. on p. 74).

Tutenel, T., Bidarra, R., Smelik, R. M., & Kraker, K. J. D. (2009). Rule-based layout solving and its application to procedural interior generation. *Proceedings of the CASA Workshop on 3D Advanced Media in Gaming and Simulation (3AMIGAS)* (cit. on p. 56).

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. 96).

Tzathas, P., Gailleton, B., Steer, P., & Cordonnier, G. (2024). Physically-based analytical erosion for fast terrain generation. *Computer Graphics Forum*. https://doi.org/10.1111/cgf.15033 (cit. on p. 33).

Vaillant, R., Barthe, L., Guennebaud, G., Cani, M. P., Rohmer, D., Wyvill, B., Gourmel, O., & Paulin, M. (2013). Implicit skinning: Real-time skin deformation with contact modeling. *ACM Transactions on Graphics*, *32*. https://doi.org/10.1145/2461912.2461960 (cit. on p. 74).

Valette, G., Herbin, M., Lucas, L., & Léonard, J. (2005). A preliminary approach of 3d simulation of soil surface degradation by rainfall. *Natural Phenomena*, 41–50 (cit. on p. 6).

Vergne, R., Vanderhaeghe, D., Chen, J., Barla, P., Granier, X., & Schlick, C. (2011). Implicit brushes for stylized line-based rendering. *Computer Graphics Forum*, *30*, 513–522. https://doi.org/10.1111/j.1467-8659.2011.01892.x (cit. on p. 4).

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. 97).

Vermeulen, T., Knopf-Lenoir, C., Villon, P., & Beckers, B. (2015). Urban layout optimization framework to maximize direct solar irradiation. *Computers, Environment and Urban Systems*, *51*, 1–12. https://doi.org/10.1016/j.compenvurbsys.2015.01.001 (cit. on p. 4).

Vila-Concejo, A., & Kench, P. (2016, August). Storms in coral reefs. Wiley Blackwell. https://doi.org/10.1002/9781118937099.ch7 (cit. on p. 67).

Villanueva, A. J., Marton, F., & Gobbetti, E. (2017). Symmetry-aware sparse voxel dags (ssvdags) for compression-domain tracing of high-resolution geometric scenes. *Journal of Computer Graphics Techniques*, *6*, 1–30 (cit. on pp. 10, 87).

Wampler, K., & Popovi, Z. (2009). Optimal gait and form for animal locomotion. *ACM Transactions on Graphics*, *28*. https://doi.org/10.1145/1531326.1531366 (cit. on p. 6).

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. 54, 58, 65).

Wojtan, C., Carlson, M., Mucha, P. J., & Turk, G. (2007). Animating corrosion and erosion. *Natural Phenomena*, 15–22 (cit. on pp. 4, 93–96).

Xing, J., Kazi, R. H., Grossman, T., Wei, L. Y., Stam, J., & Fitzmaurice, G. (2016). Energy-brushes: Interactive tools for illustrating stylized elemental dynamics. *UIST 2016 -*

Proceedings of the 29th Annual Symposium on User Interface Software and Technology, 755–766. https://doi.org/10.1145/2984511.2984585 (cit. on p. 3).

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. 95, 108).

Yersin, B., Maim, J., Pettré, J., & Thalmann, D. (2009). Crowd patches. *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 207–214. https://doi.org/10.1145/1507149.1507184 (cit. on p. 3).

Zhou, X., Benes, B., Chang, P., & Cani, M. P. R. (2021). Urban brush: Intuitive and controllable urban layout editing. *UIST 2021 - Proceedings of the 34th Annual ACM Symposium on User Interface Software and Technology*, 796–814. https://doi.org/10.1145/3472749.3474787 (cit. on p. 4).

CHAPTER A

Snake - Active Contour Model

We want to symbolize a dead coral area as a environmental object "reef." To do this, the coral objects continuously deposit a quantity of "dead coral" material. This material, stored in a discrete scalar field, contains high intensities where a reef should supposedly exist. We need to draw a curve to represent this new object. The constraints of this curve are that it must pass through the points of highest intensity while maintaining a given length.

The Snake algorithm, or Active Contour Model, approaches this application. The algorithm proposes to give an energy to the curve, which then tries to minimize it through gradient descent.

The energy, in the initial paper, is defined by

$$E^*_{\text{snake}} = \int_0^1 E_{\text{snake}}(\mathbf{v}(s))\,ds = \int_0^1 E_{\text{internal}}(\mathbf{v}(s)) + E_{\text{external}}(\mathbf{v}(s))\,ds \tag{A.1}$$

The internal energy represents the properties of the curve while the external energy represents properties of the field it lies in. In the original paper they are described as:

$$E_{\text{internal}} = \alpha E_{\text{continuity}} + \beta E_{\text{curvature}} \tag{A.2}$$

$$E_{\text{external}} = \gamma E_{\text{image}} \tag{A.3}$$

The continuity cost $E_{\text{continuity}}$, originally defined as the minimization of the spacing between the points $\left\| \frac{d\bar{v}}{ds}(s) \right\|^2$, does not make much sense in the discrete form of the algorithm. In its discrete form, we seek to maintain a regular interval between the points by applying $E_{\text{continuity}} = \left(\tilde{d} - \|p_i - p_{i-1}\|\right)^2$ with $\tilde{d}$ being the average distance between each point.

The curvature cost $E_{\text{curvature}}$ seeks to minimize the oscillations of the curve and can thus be defined as the squared second derivative $\left\| \frac{d^2\bar{v}}{ds^2}(s) \right\|^2$.

The discrete form $E^*_{\text{curvature}} = \|p_{i-1} - 2p_i + p_{i+1}\|^2$ is not necessary with splines, due to their closed form.

The image cost E_{image} tries to attract the points of the curve to a local maximum of the image gradient. It is defined as $E_{\text{image}} = -\|\nabla I\|$.

We want to see our curve maintain a given length L. We then modify the formulation of the continuity cost to become $E_{\text{continuity}} = \left(\tilde{l} - \left\|p_i - p_{p-1}\right\|\right)^2$ with $l = \frac{L}{n-1}$, knowing n is the number of vertices of the curve. Additionally, we want a curve that follows points of high intensity rather than the gradient, which leads to modifying the image cost $E_{\text{image}} = -I$.

The calculation of the gradient ∇E_{snake} remains trivial in parts:

$$\frac{\partial E_{\text{image}}}{\partial p_i} = -\nabla I(p_i) \tag{A.4}$$

$$\frac{\partial E^*_{\text{curvature}}}{\partial p_i} = -\frac{2\left(p_{i-1} - 2p_i + p_{i+1}\right)}{\left\|p_{i-1} - 2p_i + p_{i+1}\right\|} \tag{A.5}$$

$$\frac{\partial E^*_{\text{continuity}}}{\partial p_i} = 2\left(l - \left\|p_i - p_{i-1}\right\|\right) \cdot \frac{p_i - p_{i-1}}{\left\|p_i - p_{i-1}\right\|} \tag{A.6}$$

We then have

$$E_{\text{snake}} = \alpha\left(l - \left\|p_i - p_{i-1}\right\|\right)^2 + \beta\left\|p_{i-1} - 2p_i + p_{i+1}\right\|^2 - \gamma I \tag{A.7}$$

$$\nabla E_{\text{snake}} = 2\alpha\left(l - \left\|p_i - p_{i-1}\right\|\right) \cdot \frac{p_i - p_{i-1}}{\left\|p_i - p_{i-1}\right\|} - \beta\frac{2\left(p_{i-1} - 2p_i + p_{i+1}\right)}{\left\|p_{i-1} - 2p_i + p_{i+1}\right\|} - \gamma\nabla I(p_i) \tag{A.8}$$

It should be noted that the calculation of $E^*_{\text{continuity}}$ uses the distance $\|p_i - p_{i-1}\|$. For $i = 0$, the distance $\|p_i - p_{i+1}\|$ is used.

If all the points of the curve are at a distance greater than l, the optimization will push each of these points to get closer to its predecessor. Point p_0, itself, will move very little, so the entire curve aligns towards point p_0. By using the distance to the successor $\|p_i - p_{i+1}\|$, the curve moves towards point p_N. It is then possible to converge towards the median point by alternating the use of the distance with the predecessor and with the successor, at the cost of slower convergence.

The active contour model algorithm is highly sensitive to the initial curve placement. In cases where a portion of the curve is in an area with a very low gradient on E_{image}, the vertices of the curve will simply optimize E_{internal}, resulting in a straight segment in a low-intensity area, while the rest of the curve optimizes correctly.

To mitigate this problem, we propose adapting the Snake algorithm into Caterpillar: throughout the gradient descent, the target length L is artificially reduced and then increased. In this way, a portion of the curve blocked in a region without possible optimization on external energy will be attracted by the optimized curve until it falls on a strong gradient. The dead portion can take the place of optimized vertices. By returning the target length L to its initial value, the optimization continues with fewer vertices in the dead zone. Repeating this process gradually brings all points into an optimizable area. However, a too-rapid change in the target length can prevent the vertices from optimizing E_{external} by amplifying E_{internal} too much. Additionally, this algorithm can lead to numerical errors and slower convergence.

CHAPTER B

Computation of a metaball

In this section we develop in more detail the formulation used for metaballs used in Chapter 5 for the simulation of particle erosion on implicit terrains.

We use the following formula to evaluate a metaball in space with a center c and of radius R:

$$g(\mathbf{p}) = 1 - \frac{||\mathbf{p} - c||}{R}$$

using the euclidean distance.

We have a total amount Q to define in this space, so the final metaball function f needs to satisfy the equations Equation (B.1) and Equation (B.2):

$$f(\mathbf{p}) = \lambda g(\mathbf{p}) \tag{B.1}$$

$$\int_{\mathbf{p} \in V_{3D}} f \, dp = Q \tag{B.2}$$

First, let's exploit the radial symmetry of the metaball and rewrite $g(\mathbf{p}) = 1 - r$ by using the polar coordinates of the point $\mathbf{p} - c$.

We can then integrate g over the volume V_{3D} as

$$\begin{aligned} &\int_0^1 \int_0^\pi \int_0^{2\pi} g(r) r^2 \sin(\theta) \, dr \, d\theta \, d\phi \\ =& \int_0^1 \int_0^\pi \int_0^{2\pi} (1 - r) r^2 \sin(\theta) \, dr \, d\theta \, d\phi \\ =& \int_0^1 (1 - r) r^2 \, dr \times \int_0^\pi \sin\theta \, d\theta \times \int_0^{2\pi} 1 \, d\phi \end{aligned}$$

We then break down the integrals one by one such as

$$\int_0^1 (1 - r) r^2 \, dr = \frac{1}{12}$$

$$\int_0^\pi \sin\theta \, d\theta = 2$$

$$\int_0^{2\pi} 1 \, d\phi = 2\pi$$

By combining all these integrals, we get $\int g = \frac{1}{12} \times 2 \times 2\pi = \frac{\pi}{3}$.
So given $\int f = q_{\text{detachment}}$ and $\int f = \lambda \int g$, we can deduce that $\lambda = \frac{Q}{\int g} = \frac{3}{\pi} Q$.
From Equation (B.1) we finally get

$$f(\mathbf{p}) = \frac{3Q}{\pi} \left(1 - \frac{||\mathbf{p} - c||}{R}\right) \tag{B.3}$$

, representing the rate of change on the evaluation function of the terrain surface.

The integration in the voxel space is out of the scope of this paper and a numerical solution is instead proposed in Section 5.5.4.

CHAPTER C

Computation of ellipsoids in 2.5D

In the erosion process on a height field, we may want to represent the impact of a particle collision on a surface as a half sphere that we flatten to increase or lower the surface of the ground. The scaling should happen in the direction of the ground normal. In three dimensions, we may use the implicit formulation of an ellipsoid to achieve this, but in 2.5D this computation become trickier. We will first introduce the conversion to transform the 2D ellipse surface into a 1D function, then proceed with the 3D surface of an ellipsoid into a 2D function, such that it may be then used in the case of 2.5D terrain functions.

C.1 Simplified to ellipses

We will first illustrate the computation in a reduced dimension. In this case, we look at an ellipse. The common equation for an ellipse is function of x, y, the half-length a and half-width b:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{C.1}$$

We can generalize the equation to translate the center of the ellipse to the position (x_0, y_0) by setting $x' = (x - x_0)$ and $y' = (y - y_0)$ and apply a rotation θ:

$$\frac{(x' \cos\theta + y' \sin\theta)^2}{a^2} + \frac{(x' \sin\theta - y' \cos\theta)^2}{b^2} = 1 \tag{C.2}$$

For the rest of the operations, we will consider $x_0 = 0$ and $y_0 = 0$ for concisness. However this function takes x and y as parameters while we would like to remove y from the equation to lower the 2D shape in a 1D function $f(x)$.

Isolating the variable y transforms the formulation into a quadratic equation:

$$y^2 \left(\frac{\sin^2\theta}{a^2} + \frac{\cos^2\theta}{b^2} \right) + 2yx \cos\theta \sin\theta \left(\frac{1}{a^2} - \frac{1}{b^2} \right) + x^2 \left(\frac{\cos^2\theta}{a^2} + \frac{\sin^2\theta}{b^2} \right) - 1 = 0 \tag{C.3}$$

We want to solve the quadratic equation using the form

$$Ay^2 + By + C = 0 \tag{C.4}$$

Decomposing the equation Equation (C.3), we find

$$A = \frac{\sin^2\theta}{a^2} + \frac{\cos^2\theta}{b^2} \tag{C.5}$$

$$B = 2x\cos\theta\sin\theta\left(\frac{1}{a^2} - \frac{1}{b^2}\right) \tag{C.6}$$

$$C = x^2\left(\frac{\cos^2\theta}{a^2} + \frac{\sin^2\theta}{b^2}\right) - 1 \tag{C.7}$$

Solving $f(x)$ gets us to

$$f(x) = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \tag{C.8}$$

The function $f(x)$ provides us with the "upper shell" of the ellipse at the position x.

Since the function is directly related to the domain of the ellipse, we can compute the bounds of the function by identifying the two points where the tangent is vertical. From the general ellipse formulation (Equation (C.2)), we get

$$x_{\min} = -\sqrt{a^2\cos^2\theta + b^2\sin^2\theta} \tag{C.9}$$

$$x_{\max} = +\sqrt{a^2\cos^2\theta + b^2\sin^2\theta} \tag{C.10}$$

$$y_{\max} = \sqrt{a^2\sin^2\theta + b^2\cos^2\theta} \tag{C.11}$$

$$y_{\min} = \min\left(f(x_{\min}), f(x_{\max})\right) \tag{C.12}$$

The area under the curve is half the area of an ellipse, such that

$$\int_{x_{\min}}^{x_{\max}} f(x)\,dx = \frac{\pi ab}{2} \tag{C.13}$$

Finally, we consider the ground locally linear with a rotation θ such that the ground surface is defined as $g(x) = x\tan(\theta)$. So the amount of material that is being added as such can be computed as

$$\text{Area} = \int \max\left(f(x) - g(x), 0\right)\,dx \tag{C.14}$$

We correct the added area by scaling the added matter:

$$\tilde{g}(x) = g(x) + \frac{\max\left(f(x) - g(x), 0\right)}{\text{Area}}\pi ab \tag{C.15}$$

C.2 Complex case for ellipsoids

We will use a similar method to translate this idea from the ellipse to the ellipsoid.

First, the common equation of an ellipsoid is defined as x, y, z, and a, b, c respectively the half-length, half-width and half-depth of the ellipsoid:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1 \tag{C.16}$$

We can generalize the equation to translate the center of the ellipse to the position (x_0, y_0, z_0) by setting $x' = (x - x_0)$, $y' = (y - y_0)$ and $z' = (z - z_0)$ and apply a rotation (θ, ϕ). Once again, we will consider $x_0 = 0$, $y_0 = 0$ and $z_0 = 0$, but including them becomes trivial:

$$\frac{(x\cos\phi\cos\theta + y\sin\phi - z\cos\phi\sin\theta)^2}{a^2} \tag{C.17}$$

$$+\frac{(-x\sin\phi\cos\theta + y\cos\phi + z\sin\phi\sin\theta)^2}{b^2} \tag{C.18}$$

$$+\frac{(x\sin\theta + z\cos\theta)^2}{c^2} - 1 = 0 \tag{C.19}$$

We would like to remove z from the equation to lower the 3D shape in a 2D function $f(x, y)$.

Isolating the variable z transforms the formulation into a quadratic equation that we will directly decompose in the form $Az^2 + Bz + C = 0$:

$$A = \left(\frac{\cos^2\phi\sin^2\theta}{a^2} + \frac{\sin^2\phi\sin^2\theta}{b^2} + \frac{\cos^2\theta}{c^2}\right) \tag{C.20}$$

$$B = -2x\left(\frac{\cos^2\phi\cos\theta\sin\theta}{a^2} + \frac{\sin^2\phi\cos^2\theta\sin\theta}{b^2} - \frac{\sin\theta\cos\theta}{c^2}\right) \tag{C.21}$$

$$+ 2y\left(-\frac{\sin\phi\cos\phi\sin\theta}{a^2} + \frac{\cos\phi\sin\theta\sin\phi}{b^2}\right) \tag{C.22}$$

$$C = x^2\left(\frac{\cos^2\phi\cos^2\theta}{a^2} + \frac{\sin^2\phi\cos^2\theta}{b^2} + \frac{\sin^2\theta}{c^2}\right) + y^2\left(\frac{\sin^2\phi}{a^2} + \frac{\cos^2\phi}{b^2}\right) \tag{C.23}$$

Solving $f(x, y)$ gets us, once again, to

$$f(x) = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \tag{C.24}$$

The function $f(x, y)$ provides us with the "upper shell" of the ellipsoid at the position (x, y).

While the function heavily relies on the trigonometric functions cos and sin, we need to keep in mind that θ and ϕ are set for the ellipsoid, meaning that we can compute the value of $\cos\phi, \cos^2\phi, \sin\phi, \sin^2\phi, \cos\theta, \cos^2\theta, \sin\theta$ and $\sin^2\theta$ once and use them for any point (x, y).

The volume under the function is half the volume of an ellipsoid, such that

$$\int_{x_{\min}}^{x_{\max}}\int_{y_{\min}}^{y_{\max}} f(x, y)\,dx\,dy = \frac{2\pi abc}{3} \tag{C.25}$$

[TO BE CONTINUED...?]

CHAPTER D

Generation of karst networks

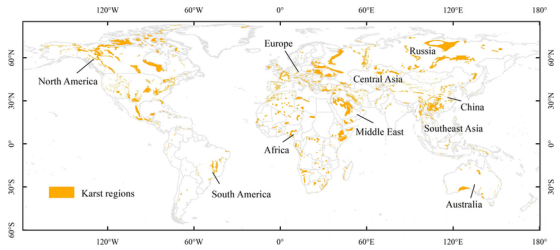


Figure D.1: Karst networks can be find anywhere on Earth. Source: Huang, Ge, et al., *Climate change and ecological engineering jointly induced vegetation greening in global karst regions from 2001 to 2020* (2022) .

Abstract

During this thesis, a novel method for the generation of karst networks have been proposed. This method is tailored by the idea of proposing fast and highly controlable methods to create geologic features. We used an adaptation of a procedural vegetation generation algorithm with very few additions. As this work does not propose an innovation but instead presents a possibility to cross the works from different research domains, we added this chapter as an appendix, such that other computer graphics works might be inspired by it.

Contents

D.1 Introduction

Karst networks are complex subterranean systems formed primarily in soluble rocks like limestone, dolomite, and gypsum. Their formation takes thousands to millions of years. Over time, groundwater erodes the rock, creating intricate, often interconnected systems of caves and conduits. We can find these formations all over the world (Figure D.1).

Karst networks offer unique ecosystems, as they are isolated from the exterior biotic and abiotic factors such as light, oxygen, etc... They contain at the same time terrestrial and underwater environments, then populated by trauglofauna and stygofauna (respectively terrestrial and aquatic animals living in complete darkness). This environment maintain stable temperatures and high humidity, providing a consistent microclimate during all seasons. This can support a range of life forms, from microbes to small vertebrates, that survive in low-energy environments.

A karst network forms through a process called chemical weathering, more specifically carbonation, where soluble rocks like limestone, dolomite, and gypsum are dissolved by slightly acidic water over long periods of time. Rainwater absorbs carbon dioxide (CO_2) from the atmosphere and from organic material in the soil, forming a weak solution of carbonic acid (H_2CO_3). Water infiltrates through joints, fractures, and bedding planes (the natural layers in the rock). Initially, these cracks are small, but as water continuously dissolves the rock along these lines of weakness, the cracks enlarge, creating conduits for water to flow more easily. Water percolates through cracks in the bedrock, it reacts with calcium carbonate in limestone, dissolving the rock in a chemical reaction. This reaction results in calcium ions (Ca^{2+}) and bicarbonate ions (HCO_3^-) being carried away by groundwater, slowly enlarging the cracks and fractures in the rock. As the water continues to dissolve rock along these pathways, small cracks can grow into large underground channels or tunnels, which are the beginnings of cave systems. Over time, these underground channels expand into larger voids, forming caves.

Many factors influence the speed and size of karst formations. The main factors are:

- The rock composition, which can have different concentration [CHECK THIS TERM] of $CaCO_3$, reacting more or less with acidic water and thus resisting proportionnaly to the erosion.
- Fractured bedrock facilitate the water pathway, easing the weathering process in a certain direction.
- Surface vegetation density also plays an important role as it produces CO_2 that is being drained by rain water, accelerating the carbonation of the bedrock. Precipitation and atmospheric humidity increase the amount of infiltrated water. These environment parameters are significantly correlated (Bari et al., 2021).

Karst systems have an important role in the environment as they enable the storage and purification of ground water while providing a continuous water flow at the surface. This feature have a large impact for the surface environment, as it serves as aquifer and drought buffer, which is useful for the surface ecosystems or human agriculture.

Karst subterrains can be seen as networks of caves connected through channels with sizes that varies greatly (Collon, Bernasconi, et al., 2017; 2021). The underground study is a

difficult task as the channels can become too small for a human, and can be fully submerged for an undefined distance, making spelaeological expeditions dangerous. Important technologies have arise such as LIDAR for the 3D modeling of explored caves and underground-GPS systems for studying the system's topology (Collon, Bernasconi, et al., 2017).

D.2 Related works

- Modeling based on geology (no control):
** (Jaquet et al., 2004; Pardo-Igúzquiza et al., 2012) - State of the art:
** (Paris, Guérin, et al., 2021), more recently (Gouy et al., 2024)
*** 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)
*** Plus, the computation of sampling and cost graphs can be very expensive, reducing possibilities of user interaction. *** Fractures typically form due to changes in overlying pressure or tectonic stress from faulting or folding. To gain a more comprehensive understanding of the location, orientation, and extent of fractures in rock, as well as their hydrogeologic behavior, geophysical surveys can be useful. These investigations can employ a variety of geophysical tools, including surface geophysical instruments (non-invasive) and downhole geophysical logging, which requires a borehole. However, numerical simulation of bedrock fractures is highly complex and demands a deep understanding of the surrounding environment to achieve accurate results.
** (Pytel and Mann, 2015)
*** Based on voxels, looks plausible, with a low number of parameters, but take way too long (1̃0 to 20 minutes per generation)
*** Which makes it unfit for user interaction
** (Collon, Steckiewicz-Laurent, et al., 2015; Collon, Bernasconi, et al., 2017; Jouves et al., 2017)
- With some imagination, we can see the shape of trees:
** [Prusinkiewicz : ANY]
** (Runions, 2008)
- Or miscelleanous networks:
** (Galin, Peytavie, et al., 2010; Fernandes and Fernandes, 2018)
- Maybe even look for lychen or ant colonies
- ...

D.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]
- ...

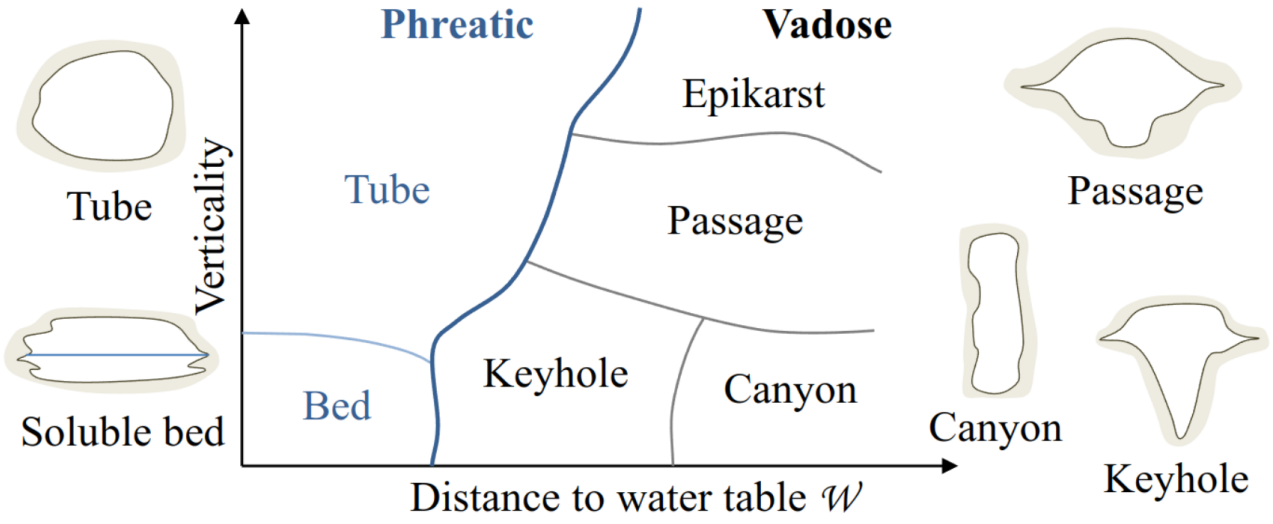


Figure D.2: Classification of tunnel shapes

D.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
- ...

D.5 Modeling

- Tunnels:
** The output of SC is a set of paths
** (Paris, Guérin, et al., 2021) provides a classification of tunnel shapes (Figure D.2).
- ...

D.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
- ...

D.7 Results

- ...

D.8 Conclusion

- ...