

**M A S A R Y K
U N I V E R S I T Y**

FACULTY OF INFORMATICS

**Procedural Generation of Riverbeds
in Enfusion Engine**

Master's Thesis

BC. JAKUB IVANICKÝ

Brno, Fall 2022

**M A S A R Y K
U N I V E R S I T Y**

FACULTY OF INFORMATICS

Procedural Generation of Riverbeds in Enfusion Engine

Master's Thesis

BC. JAKUB IVANICKÝ

Advisor: Mgr. Jiří Chmelík, Ph.D.

Department of Visual Computing

Brno, Fall 2022



Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Jakub Ivanický

Advisor: Mgr. Jiří Chmelík, Ph.D.

Acknowledgements

I want to thank to my supervisor, Mgr. Jiří Chmelík, Ph.D., and to my advisor from Bohemia Interactive, Ing. Ivan Buchta, for suggesting this intriguing topic and for all the time and effort they put into leading me to make it into a master's thesis. I would also like to thank to my colleagues from Bohemia Interactive for their valuable input on the tool, especially Jan Trecha and Ondřej Branžovský from the environment design team for their guidance and feedback, and Lukáš Závitkovský, Pavel Šafář, Peter Čimo, and Filip Doksanský from the Enfusion team for their technical support regarding the implementation. Finally, I am grateful to my family and friends, who have immensely supported me in my studies despite me spending most of the last year annoying them by complaining about how exhausting writing a master's thesis is.

Abstract

Hydraulic erosion is one of the most important landmass-defining features in Earth's geology. While many techniques are available for the procedural generation of virtual terrains, including simulation of various types of erosions, these methods are not suitable for the generation of stable river channels. This thesis proposes a tool for the procedural generation of rivers, where the user only defines the river's source. The river course and its parameters are then generated, and the channel is carved into an existing terrain. Methods for generating meandering, braiding, and anastomosing rivers are also discussed.

Keywords

procedural terrain generation, topology, rivers, riverbeds, morphology, digital elevation modelling

Contents

Introduction	4
Objectives of the thesis	6
1 Generating terrains and rivers	10
1.1 Terrains in video games	10
1.1.1 Terrain generation software	11
1.1.2 Creating terrains using noise	12
1.1.3 Simulating erosion	17
1.2 Rivers, their behaviour and classification	21
1.3 River generation and simulation	31
1.3.1 Generating river networks	31
1.3.2 Generating rivers by modelling riverbeds	34
2 Proposed method	37
2.1 River representation	37
2.2 The optimal rivercourse	39
2.3 River cross section parameters	43
2.4 Evaluating the terrain changes	50
2.4.1 Meandering rivers	55
2.4.2 Braiding and anastomosing rivers	58
3 Implementation in Enfusion Engine	61
3.1 Terrain generation in Enfusion Engine	61
3.2 Applying the terrain changes	63
3.3 Performance analysis and optimization	65
3.3.1 Regenerating rivers after changes	68
4 Discussion	71
4.1 Shortcomings and possible improvements	71
4.1.1 User experience	71
4.1.2 Generating optimal rivercourse	72
4.1.3 Meanders	74
4.2 Further work	75
4.2.1 Cascading	75
4.2.2 Confluences	76

4.2.3	River surface and flow maps	77
4.2.4	Terrain textures and vegetation	78
Conclusion		79
Bibliography		81
A	River generator manual	86
A.1	Starting the Workbench and loading a map	86
A.2	Adding a new river	89
A.3	Generating the rivercourse	89
A.4	Generating the parameters	91
A.5	Setting river parameters manually	93
A.6	Applying the river generators	93
A.7	Creating a custom map	96
B	List of third-party code	97
C	Electronic attachments	98
D	Screenshots	99

Glossary

Braiding and anastomosing rivers are rivers formed of multiple intertwining channels. They usually occur in areas where large amounts of sediment are deposited by the river.

Erosion is a process in which soil, rocks, or other sediment is transported due to physical and chemical forces occurring in nature.

Game environment, game world, game landscape are, within the context of this thesis, exterior levels of a video game.

Heightmap is a form of digital representation of a terrain elevation by sampling it at regular intervals. The size of the intervals is known as a heightmap resolution.

Hydraulic erosion is a process in which sediment is transported by water flowing on the terrain's surface.

Meander represents one of a series of curves created on a rivercourse by erosion of the outer bank and sediment deposition in the inner bank of the river.

PointData is a structure containing multiple parameters that is assigned to a spline control point, and can be used by any object to store data that can then influence its behavior. See page 38.

Polyline is a piecewise linear function composed of straight-line segments fitting through a set of points.

Rivercourse, watercourse is a channel followed by a stream of water. In this thesis, the word is mostly used to describe the shape of the main channel of the river in the terrain that can be defined by a spline shape.

RiverGeneratorEntity is a class implemented in the script, that inherits from `SplineTerrainGeneratorEntity` and implements all high-level features of the river generator tool.

RiverGeneratorPointData is a derived class of `PointData` used by the `RiverGeneratorEntity` to store river cross-section and planar parameters at spline control points. See page 38.

Sinoid meander represents a meander which oscillates around the mean river direction, without ever flowing against the valley direction. See Figure 2.4.

Spline is a parametric piecewise curve, which can be used to describe complex shapes. In this thesis, *spline* refers to the entire curve and *spline segments* refer to the separate pieces.

Spline tessellated points are points that are temporarily created when a spline is evaluated to reduce it into a polyline while maintaining its smoothness.

Spline distance is a distance along a spline between two points it passes through.

Spline control points are points that describe the spline and separate it into multiple segments (pieces). In *Enfusion Engine*, each control point can be assigned a set of `PointData` objects, which can influence the properties of objects represented by the spline.

SplineEntity is a type of an object in the *Enfusion Engine*, which is used in the *World Editor* to manipulate a spline.

SplineTerrainGeneratorEntity is a class implemented in the game engine, that provides generic API for various type of terrain generators.

tcp(tp) is a function used multiple times during the generating process which, for a specified tessellated point, returns the index of its' preceding control point. If a tessellated point is the same as a control point, the index of the control point is returned (as opposed to the strictly preceding control point). See page 47.

Terrain is a representation of the ground in an exterior level of a video game. Terrains usually only represent the soil, with objects such as large boulders being modeled as separate objects.

Thermal weathering is a process in which minerals deteriorate and break down due to contact with water, chemicals, or biological organisms. Unlike erosion, thermal weathering usually does not lead to transportation of the matter over large distances.

Tightly-looped meander represents a meander where a curve is pronounced enough to make the river temporarily flow against its mean direction. See Figure 2.4.

Introduction

Ever since the rise of the first video games, visual fidelity and environment realism have been one of the main, if not the main, aspects in which both hardware manufacturers and game development studios keep trying to surpass their competitors. Whereas the first games would, due to limited resources, usually only simulate objects essential for gameplay purposes, improvements in terms of both hardware and software soon allowed them to create simple environments that would visually mimic the real world. Thanks to the exponential growth in hardware performance, as well as the capacity and speed of storage devices, games of recent years often do not have to limit themselves to small-scale "levels" as their predecessors. Instead, it is possible to recreate massive yet detailed environments for players to explore. One of the effects of this was the emergence of the so-called open-world games, which present the player with environments consisting of entire cities or regions, albeit usually at reduced dimensions.

However, including such large and details environments presents more problems than just their simulating and rendering. While some games generate their worlds (or, in many cases, whole universes) entirely procedurally - see, for instance, *Minecraft* or *No Man's Sky* - many other games require this process to be done manually by the game developers' environment teams. This requires modeling the topography, hydrology, climate, vegetation, settlements, and infrastructure while having to keep in mind the relations that form between them. Since players keep expecting larger and more realistic environments, the work required to design them keeps increasing fast. For this reason, procedural terrain generation has been becoming increasingly useful, even for games that take place on otherwise hand-designed terrains.

While multiple (usually commercial) software solutions for solving some of the above-mentioned aspects of the environment design exist, they focus mainly on modeling the overall topography or generating vegetation. Modeling riverbeds is usually performed using sculpting tools provided by the game engine, editing the heightmap in an image editor, or other manual methods. In an attempt to bridge this gap, this thesis presents a novel way of generating rivers in vir-

tual environments. The method and implementation were designed after consultations with the environment design team of Bohemia Interactive in an attempt to create a tool that would directly address those aspects of river generation that they consider most suitable for automation. The implementation of the proposed method was done in Bohemia Interactive's proprietary Enfusion Engine¹, which is designed with an emphasis on the development of open-world titles. However, the proposed method and most implementation aspects should be easily transferable to any game engine capable of working with splines and terrain heightmaps.

The first chapter of this thesis begins with an overview of previous works on terrain generation, including large-scale simulations of landmass evolution processes. I also describe some of the programs available for the production of the initial landmass by simulating processes like erosion, discussing their uses as well as limitations. The chapter then examines rivers, referring to existing works that describe the rules of their formation and the properties they exhibit to the extent useful for our use case. Lastly, it describes previously presented methods of generating rivers in virtual environments.

The second chapter focuses on the novel method proposed by this thesis. First, the representation of rivers in the tool is discussed. I then divide the generation into three steps - rivercourse generation, parameters generation, and applying the river to the terrain heightmap. Special attention is given to meandering, braiding, and anastomosing rivers, which many previous works have overlooked.

The third chapter focuses on the reference implementation in Enfusion Engine. The first section describes important specifics of the engine that influenced the implementation. I then describe the approach to the effective application of the terrain heightmap modifications. The last section focuses on the method's performance and describes suitable optimizations.

The fourth chapter discusses some of the shortcomings of the proposed and implemented method and attempts to introduce possible solutions. It also describes potential directions for further work that would make use of the immense potential of procedural river generation.

1. <https://enfusionengine.com/>

The implemented tool is part of the Enfusion Workbench toolset used for developing games in the Enfusion Engine and was evaluated by Bohemia Interactive's environment design department. At the time of publication, Workbench is only distributed together with the game Arma Reforger. However, a build of Workbench with limited functionality is available for the purpose of testing the proposed tool, and is included in the thesis archive².

Objectives of the thesis

The main objective of the thesis is to design and implement a tool for carving riverbeds to video game terrains that game environment designers can use within the game engine's map editor. The current method of manually carving rivers and streams requires designers to sculpt the landscape by hand. This not only leads to often inconsistent results (from minor issues, such as unrealistic changes in the river dimensions, to major ones, such as rivers flowing uphill) but is also highly time-consuming for more expansive environments. Instead, designers could use an automated tool to make the required terrain adjustments while ensuring that the results follow basic hydrological laws.

I have developed the thesis in cooperation with the environment design department from the game development company Bohemia Interactive, and the implementation of the tool was designed based on their requirements and expectations:

1. **The tool will be available within the World Editor tool of Bohemia interactive's Enfusion Engine.** Like most commercial game engines, Enfusion Engine also allows users to create and modify levels using a dedicated tool similar to those used by engines such as Unity or Unreal. Similarly, Enfusion enables the user to create terrains in their exterior levels³ and modify them in the

2. Available at: <https://is.muni.cz/auth/th/dss2n/>

3. In video game engines, there are technical limits to the maximum dimensions of game objects, which are often not big enough to be able to encompass the whole open-world levels that those games contain. However, terrains are game objects with limits put on their geometry that allow them to be optimized enough to represent the entire landmasses while ensuring a manageable simulation and rendering performance.

World Editor using basic sculpting tools - raising, lowering, and smoothing the terrain. All features of the river generator tool will also be integrated into the World Editor, so that designers do not have to export and import terrain data to and from the engine.

2. **The tool will be able to propose possible river layouts, set realistic dimensions and other parameters for the river, and then carve the river into the terrain.** To minimize the amount of workload expected from the environment designer, they are only required to set the river's source. The tool will then start by generating the river course beginning at this point by sampling the terrain and extending the river downhill until it either reaches an elevation target set by the user or reaches a local minimum (a point from which the river would have to flow uphill, no matter the direction). Then, the tool will generate parameters of the river - its dimensions, the shape of the riverbed and banks, the size of the river's meanders, and the number of the channels the river forms⁴. Finally, the shape of the riverbed will be carved into the terrain and can be filled with a river surface.
3. **The user has to be able to modify the outcome of the tool at every step of the generation.** Since in most games, rivers are not only a visual feature but also present a gameplay element by impeding or expanding movement options, environment designers might want to adjust the generated rivers. As such, the tool must allow them to change the route of the river and its parameters without forcing them to use manual sculpting tools.
4. **The user must be able to supply their own data for some steps of the river generation.** Since open-world maps are often created based on real-world locations, environment designers can use available map data to add rivers to the virtual terrain - those are often imported as spline data. In such a case, the tool should be able to accept those spline data and still generate fitting river parameters. At the same time, designers might use the tool to generate the river course but set their custom river parameters.

4. For further description on rivers, their classification, and properties, see Section 1.2.

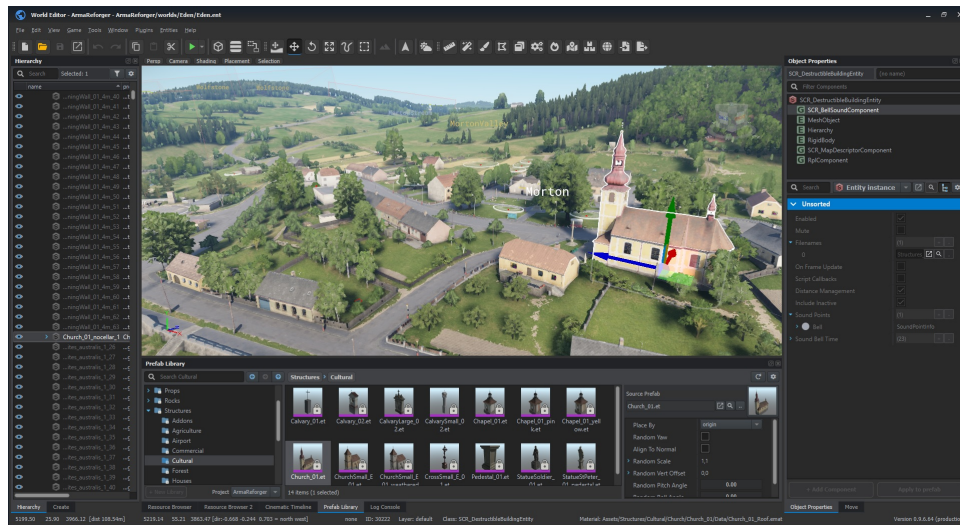


Figure 1: World Editor, a tool for creating and modifying environments for games using Infusion Engine.

5. **The tool should be expandable and modifiable by the modding community.** Like many other engines, Infusion Engine allows the players to use a set of tools to modify the game. This includes creating new maps using the World Editor, modifying the game logic using the Script Editor, or adding animations to the in-game characters using the Animation Editor. Since the engine allows scripts written in the Script Editor to be executed not only in the game but also in the World Editor, users should be able to add or modify features in the river generator tool. For instance, the user might want to expand the tool so that it populates the river's surroundings with vegetation and rocks. This should be possible from within the script.
6. **The tool should primarily focus on larger rivers instead of smaller streams.** Since the resolution of a heightmap representing the terrain is limited, it is often challenging to design small streams while ensuring they look realistic. While it is something an expert designer can still manage, many games omit very small streams altogether. The tool should therefore focus on generating wider rivers, as the results of generating small streams

could be too inconsistent and would have to be still manually checked and adjusted by the designers.

As such, the next chapter will evaluate available solutions for generating terrains and rivers according to those requirements.

1 Generating terrains and rivers

In this chapter, I provide an overview of some of the methods for terrain generation from scratch that are presently being used in video game development. Since this thesis exclusively focuses on generating rivers in existing environments, the following section describes rivers from the geographical and hydrological points of view, highlighting the features of rivers that we will try to reproduce in the presented method. The final section then examines related works that also focus specifically on adding riverbeds to existing terrains.

1.1 Terrains in video games

Virtual terrains are commonly represented in one of two ways - as a regular grid of elevation samples, also known as a heightmap, or as an irregular triangle network (triangle mesh), often also used for representing any other visual objects. While triangle meshes consist of vertices positioned anywhere in a 3D space, with surfaces always defined by a triplet of points, a heightmap only allows for one elevation sample per one 2D coordinate. Additionally, since they represent a regular grid, the spacing between the coordinates at a heightmap has to be constant. As a result, triangle meshes can describe a terrain of virtually any shape. In contrast, heightmaps do not support terrains that contain overhangs and caves since those would require multiple vertical samples per one 2D coordinate. Additionally, heightmaps only offer a constant resolution of the terrain for all biomes and topological features - mountain peaks and riverbeds are forced to have the same level of detail as valleys and planes, despite the elevation variety found in these environments, and thus the elevation sampling precision useful for describing them, being widely different. On the other side, triangle meshes can have high vertex density in areas where detail is needed, while areas, where the terrain is uniform and contains no details can be represented with a lower vertex count.

However, the uniformity of the heightmap makes them not only easier to store (as they can be defined only by their dimensions and an array of elevations) but also makes it possible to manipulate them as images since the elevation data can be visualized and processed as

a single color channel. This allows terrain generation software to use simple image processing functions and algorithms, such as various noises, blurs, and brightness and contrast adjustments, to easily generate and modify heightmaps to create realistically looking terrains. Game engines can easily simulate and render heightmap-represented terrains as a regularly subdivided 2D plane with various vertical vertex offsets, simplifying collision detection and physics simulation, as well as view culling and rendering. [1]

For these reasons, heightmaps are currently used by both most of the terrain generating software, as well as the default terrain implementation in most computer games engines [2, p. 18], including Enfusion Engine. For this reason, the rest of the thesis always describes terrain processing in terms of heightmap modifications.

1.1.1 Terrain generation software

Undoubtedly, video game environments' design has a massive impact on the player's experience. While games focusing on a more limited narrative usually utilize linear levels, where the game designers have greater control over the gameplay, games that provide a more emergent experience often place the player in larger maps, where they have multiple ways of completing the game's objective. However, larger maps require game developers to spend more time and effort to create both captivating visuals and offer a certain level of gameplay use. Moreover, with the technical limitations being constantly pushed by the more powerful hardware, players expect games to make use of this by presenting them with larger worlds that are more detailed and realistic than before. This is most noticeable when we look at modern open-world games, which recreate entire regions that the players can traverse as their wish.

While those environments were often created in the past by sculpting the terrain and adding objects by hand, today's massive open worlds require the environment designers to offload some of the necessary work to procedural generation. Since most games create environments that, to some extent, follow natural rules present on Earth, examining real geographical phenomena and simulating them can often create believable worlds, requiring the environment artists to expend much less time creating them.

Like in reality, the general topography of a map often greatly influences the layout of the open world itself - the position of mountains and valleys dictates the locations of settlements and the infrastructure that connects them. Being the result of millions of years of geological processes, the terrain's topography and geology are incredibly varied and detailed. On one side, this makes them demanding to be reproduced by hand. However, knowing the rules that drive these geological processes makes it possible to generate terrains by simulating them virtually.

As such, multiple software products give environment artists the ability to quickly define a general layout of a map, including mountains, valleys, and ridges. While many different tools use manual sculpting, fewer offer the user the option to simulate geological processes and automatically create finer landscape features. Some of the notable ones are World Machine¹, World Creator², Instant Terra³, Terragen⁴ and Gaea⁵.

As I have mentioned at the chapter's beginning, all those programs represent the terrain as a heightmap. As such, they allow the user to either import an existing dataset (for example, heightmaps of real-world locations are available for free from the U.S. Geological Survey EarthExplorer databases⁶), or to create a new one from scratch.

1.1.2 Creating terrains using noise

Fractal noises have been employed by the terrain generators to simulate the variability of terrain found in nature since the introduction of fractional Brownian motion by Mandelbrot; Van Ness [4]. Fractional Brownian motion (fBm) combines multiple layers of underlying noise samples, starting with high amplitude and small frequency (which creates more prominent features, such as mountains) and then exponentially decreasing the amplitude and increasing the frequency for each new layer added (creating larger rock formations at first, and

1. <https://www.world-machine.com/>

2. <https://www.world-creator.com/>

3. <https://www.wysilab.com/>

4. <http://planet-side.co.uk/terrigen-overview/>

5. <https://quadspinner.com/>

6. <https://earthexplorer.usgs.gov/>

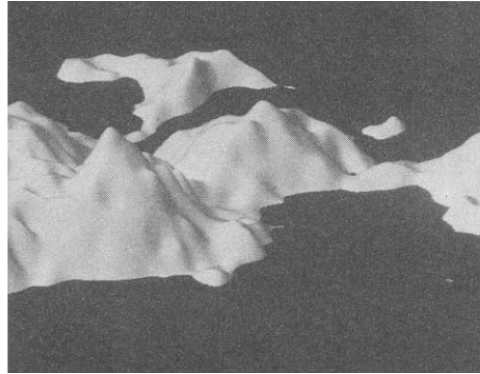


Figure 1.1: Surface produced by Fournier et al. [3] using fractional Brownian motion in 1982.

separate rocks at the end). Fournier et al. [3] applied fBm to generate mountainous terrains, a height map of a planet, as well as to add detail to a simplified shape of a continent. Perlin [5, p. 289] presented a novel type of noise, known as Perlin noise, which for a n -dimensional noise defines a n -dimensional grid with a n -dimensional pseudo-random unit-length gradient vector assigned to each grid intersection. The gradient vectors are generated using a permutation lattice, which contains uniformly distributed numbers 0-255. For any queried point, Perlin noise determines the grid cell in which it is located and its position within that cell in the form of offset vectors to each of the cell's corners. For each corner, it then calculates a dot product between the gradient vector and the offset, with the final value of the noise being the interpolation of those dot products based on the distance from the corners of the grid cell. Compared to the previously used value noise (where random values are generated for each grid intersection and then interpolated between based on the position of the point in the grid cell), the use of the dot product, which forces the points surrounding the grid intersection point to include all values in the $[-1, 1]$ range, prevents the noise from creating larger areas with nearly constant values. Combining the two using the fractal noise technique or by simple blending is also possible and can be used depending on the nature of the generated terrain. Adding masks over the noise layers can create areas with a higher or lower elevation variation, ranging from mountains (with a strong influence of both high- and low-frequency noise),

through hills (with a low influence of high-frequency noise) and to rolling hills and plains/plateaus (with no influence of high-frequency noise and low effects of low-frequency noise).

By applying basic numerical transformations to the result, we can further modify the result: an absolute value of the result will create a terrain with sharp V-shaped valleys and rounder mountains, in terrain generation sometimes called *billowy noise*. Inverting billowy noise, on the other hand, creates terrain full of sharp ridges over rounder valleys, sometimes called *ridged noise*. Terraces can be created by applying a smoothstep function within certain intervals of the elevation range, gaussian blur can be used to smoothen out sharp small features, and the whole elevation range can be clamped or multiplied to add or remove vertical variety. To combine those transformations, terrain generation software usually uses graphs to define terrain generation behavior.

While noise appears to be the most common way of generating rugged terrain, some authors have combined it with other methods or proposed entirely different ways of achieving similar results. Kamal; Uddin [6] propose a method where a user specifies a position of a mountain and the extent of its base region. The algorithm then, throughout multiple iterations, repeatedly generates random cutting lines to split the terrain into polygons. It then selects the polygon containing the mountain peak and a few surrounding polygons (by performing a random walk of a constant length). Finally, it raises the terrain in all selected polygons by a constant amount.

Olsen [7] proposes a method that combines $1/f$ noise (a variant of fractal noise) with Voronoi diagrams⁷ by using them as a mask to create isolated mountains separated by valleys. Separate polygons of the Voronoi diagram can then be assigned random fractions in the 0-1 range to specify the variation found in the polygon, with 0 creating flats, 1 mountains, and intermediate values hills of varying heights.

Finally, Ong et al. [8] propose a method of generating terrain using genetic algorithms, where the user inputs a *silhouette map* describing a layout of different types of terrains (for instance, a region of

7. A plane partition into n polygons by introducing n points into the plane at random positions and then dividing the plane so that every point in the polygon is closer to the generating point than to any other point, <https://mathworld.wolfram.com/VoronoiDiagram.html>.

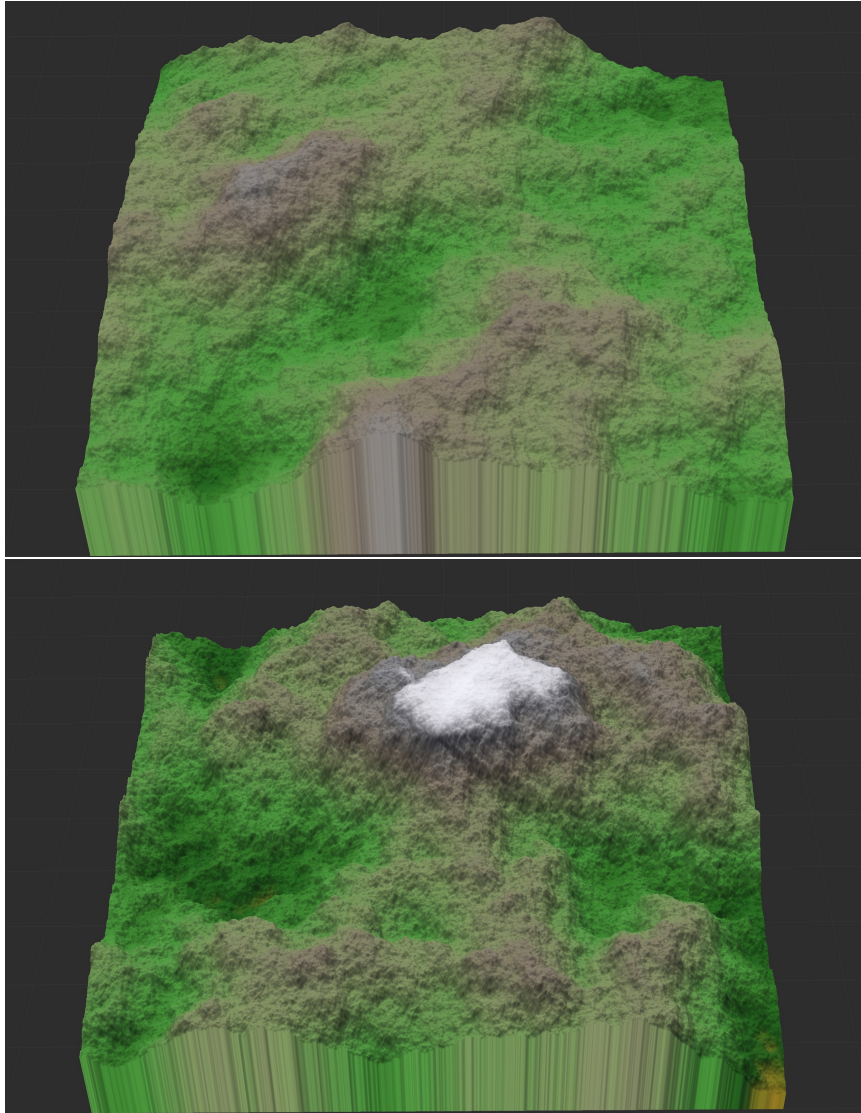


Figure 1.2: A visualization of terrains produced by Perlin noise (top) and Value noise (bottom) in World Machine at the same scale. Notice that the value noise creates a mostly flat mountain in the upper center section of the terrain and a large basin in the middle left section. On the other hand, the Perlin noise terrain has more constant variability over its entire size.

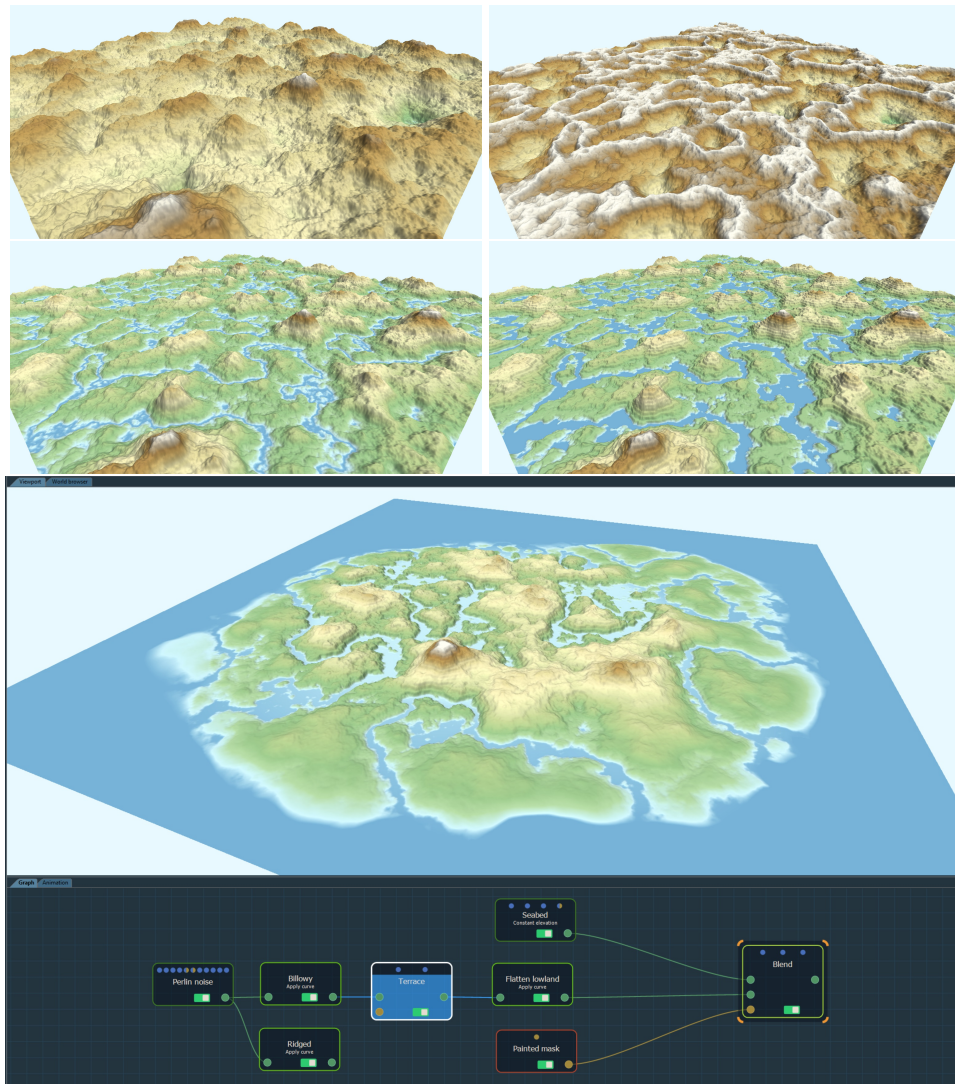


Figure 1.3: Perlin noise terrain (top left) with ridged (top right) and billowy (middle left) variants. Thanks to the graphs interface, users can quickly manipulate the terrain further, such as by adding terraces to the billowy terrain (middle right), smoothing it, and adding a mask (bottom).

mountains surrounded by hills further surrounded by plains) and a set of existing elevation data samples for each of those types. The algorithm then creates a grid of *genes*, where each gene represents a point in the heightmap and is assigned a random terrain sample from the corresponding data set. To resolve blending issues, the genes are allowed to overlap slightly, and the final terrain is blended in those overlapping regions. Then, the algorithm generates *chromosomes* - sequences of image transformations, such as translation, rotation, and scaling of the elevation data. *Mutations* and *crossovers* are then performed on the chromosomes repeatedly - separate transformations can be switched for different ones, and parts of the sequences can be transferred between the chromosomes. At the end of the evolution process, the fitness of chromosomes is evaluated based on the similarity of the generated data compared to the input samples to pick the best-fitting one, and all the operations carried by that chromosome are then applied to the initial heightmap carried by the genes.

1.1.3 Simulating erosion

While noises were commonly used to create the first procedurally generated terrains, they are insufficient to create realistically looking topographies by themselves. Unlike those simple "mountains", Earth's mountainous regions have been smoothed out by millions of years of erosion, moving mass from sloped areas to flat ones, creating valleys, alluvial fans⁸, terraces and deltas, whose shape is often smoother than that of the mountain tops [9, p. 41]. For this reason, more advanced terrain-generating software allows the user to simulate various types of erosion on top of the noise-generated terrain. In geological fields, methods for simulating erosion take into account the soil's physical properties, detailed terrain roughness, and the realistic amount of rainfall or exposure to wind and simulate the erosion over multiple iterations. However, methods for generating terrains for video games often omit these details, using more generic formulas while usually still making use of the iterative approach.

8. A deposit of sediment in the shape of an inverted funnel, commonly found at the foothills - <https://education.nationalgeographic.org/resource/alluvial-fan>.

Musgrave et al. [9, p. 41] were one of the first to present a simplified erosion model, which simulated the dissolution, transportation, and deposition of terrain by two forces, thermal weathering as well as hydraulic erosion, over a terrain represented by a mesh. Thermal weathering moves the soil downhill at all vertices with an incline sharper than a user-defined *talus angle* constant, moving the incline itself up in the process and creating a sediment fan at the bottom of it. Eventually, this leads to the incline of the topmost part of the slope approaching the talus angle with a transition towards a flat surface at the bottom. The hydraulic erosion model functions as follows [9, p. 45]:

1. For each heightmap vertex v , let us have ground elevation a_v , amount of water currently on the vertex w_v , and the amount of sediment stored in the water s_v . $K_c = 5.0$ is a sediment capacity constant determining the maximum amount of sediment in a unit of water, and $K_d = 0.1$ and $K_s = 0.3$ are the depositing and dissolving constants determining the speed of soil deposition and dissolution.
2. For each pair of neighboring vertices (v, u) , calculate the passing of water Δw (from point v to point u) by comparing the difference between their water levels:

$$\Delta w = \min(w_v, (a_v + w_v) - (a_u + w_u)). \quad (1.1)$$

3. For the vertex v with the lower water level, eg. the one with $\Delta w \leq 0$, some of the sediment s_v stored in the water already present at v now gets deposited at v :

$$a'_v = a_v + K_d * s_v \quad \text{and} \quad s'_v = (1 - K_d) * s_v, \quad (1.2)$$

as the water will stay at the vertex until the next iteration.

4. For the vertex v with the higher water level, eg. the one with $\Delta w > 0$, some of the water moves into the neighboring point:

$$w'_v = w_v - \Delta w \quad \text{and} \quad w'_u = w_u + \Delta w \quad (1.3)$$

Also, the *sediment capacity* $s_{cap|v}$ of Δw is calculated as $s_{cap|v} = K_c * \Delta w$.

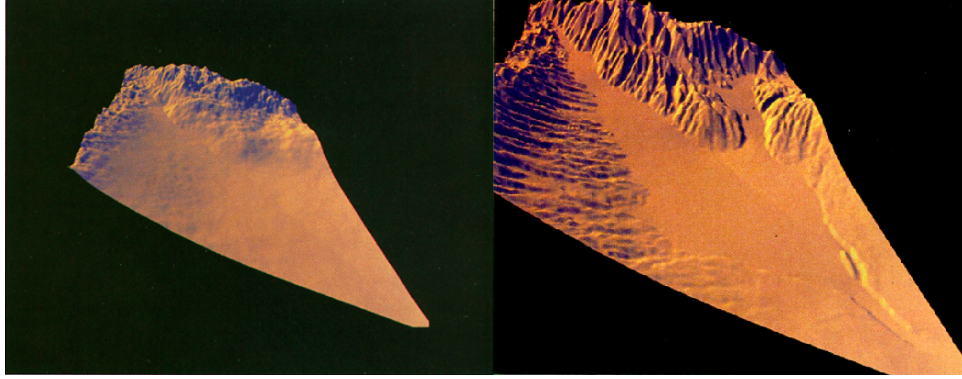


Figure 1.4: A patch of terrain after 2000 iterations of Musgrave et al. [9]’s hydraulic erosion and thermal weathering.

- (a) If the amount of sediment s_v is higher than the permitted $s_{cap|v}$, then any sediment that can be moved to u is transported: $s'_u = s_u + s_{cap|v}$, while some of the remaining sediment gets deposited:

$$a'_v = a_v + K_d * (s_v - s_{cap|v}), \quad (1.4)$$

$$s'_v = (1 - K_d) * (s_v - s_{cap|v}). \quad (1.5)$$

- (b) If, on the other hand, the water flow is able to carry the sediment to the neighboring vertex ($s_{cap|v} \leq s_v$), then transport not only all the already dissolved sediment, but also some of the soil from v , while no sediment remains at v :

$$s'_u = s_u + s_v + K_s * (s_{cap|v} - s_v), \quad (1.6)$$

$$a'_v = a_v - K_s * (s_{cap|v} - s_v), \quad (1.7)$$

$$s'_v = 0. \quad (1.8)$$

5. At regular intervals (60-100 iterations), raise w_v for all points by a set amount K_r to simulate rain.

While the model is only a very simplified description of natural erosion, it does not require the user to input any additional data, such as the soil composition or rainfall volume, making it helpful in

designing terrains for video games or other entertainment products. Some authors have chosen to expand or modify this method - Beneš; Forsbach [10] added evaporation to the existing algorithm. Olsen [7, p. 10] improved the method's execution time by reducing the number of cell neighbors from 8 to 4, simplified the water distribution process, and assumed a constant sediment content in the water instead of maintaining a sediment map. Anh et al. [11] have implemented the algorithm on the GPU instead of CPU, leading to a decrease in the execution time of the observed cases by 80-92%.

Chiba et al. [12] proposed a method that uses velocity fields for simulating the water movement in a continuous way that respects the inertia of the moving water by adding a map of velocity vectors and collision energy. The flow is modeled by simulating the motion of separate water particles, where acceleration (calculated from the height difference and gravitational acceleration), velocity, and energy (calculated from the difference between the collision tangent and the particle movement) are calculated per particle and the energy applied to determine the intensity of the erosion.

Jákó; Tóth [13] combine the Musgrave et al. method with the velocity fields concept and further improve it by simulating additional phenomena, such as the depth of the water affecting its speed (since deeper water often moves slower), an improved 3D collision between water and soil (where erosion is stronger with the angle between water motion and soil being closer to perpendicular) and gradual softening of the soil after prolonged exposure to the erosion. This method was also implemented on the GPU to decrease the time needed for simulation, together with a modified parallelized thermal weathering algorithm.

A more realistic method is also presented and implemented by Beneš et al. [14]. Unlike the ones mentioned above, it uses Navier-Stokes equations to compute the movement of the fluid more realistically and simulates the force exerted on the soil by the moving water in an environment represented by a triangle mesh. For this reason, this method can take into account the varying density and viscosity of the liquid, as well as different soil particle sizes, soil material frictions, and different waterbed geometry. The more precise fluid simulation allows the algorithm to reproduce better phenomena present in the real world, such as a flood eroding meanders and creating oxbow

lakes⁹, while using a triangle mesh instead of a heightmap allows it to be simulated on more complex geometries than terrains.

Finally, Štava et al. [15] propose a method combining the Musgrave et al. method with the possibility to simulate multi-layer terrains consisting of multiple soil types and water penetration into the deeper soil levels. Furthermore, the terrain is divided into tiles that are later combined so that even larger terrains that would not fit into GPU memory can be simulated on the GPU.

While the commercial terrain generation programs do not directly acknowledge which erosion algorithms they implement, the set of parameters they expose to the user suggests that each includes a custom implementation based on the abovementioned methods. The user can specify the amount of rain, the erosive power of the flowing water, and in some cases, the influence of evaporation (World Creator) or a specific water viscosity (Instant Terra). Instant Terra offers multiple filters to separate different forms of erosion, such as thermal weathering or hydraulic erosion. In contrast, World Machine allows the user to select between *Standard Erosion* and *Channeling Erosion*, representing weathering and hydraulic erosion, respectively. World Creator also allows the user to specify the wind's direction and strength, thus having more control over the thermal weathering.

Instead of generating terrains during development, many games use compute shaders to generate terrains on the graphics card during runtime. Space simulation games, such as No Man's Sky or Elite: Dangerous, use this method to include realistically scaled planets while requiring only a minimal set of parameters to be stored on the user's storage device.

1.2 Rivers, their behaviour and classification

While the methods of simulating hydraulic and other types of erosion in the previous section provide interesting and visually pleasing results, their use for creating rivers, which is the focus of this thesis, is limited [16, p. 57][17, p. 2]. The control over the river-forming process that they lend to the user is almost none, and due to the limitations of the simulation, they usually only lead to the formation of valleys,

9. A U-shaped lake that forms when a meander is cut off.

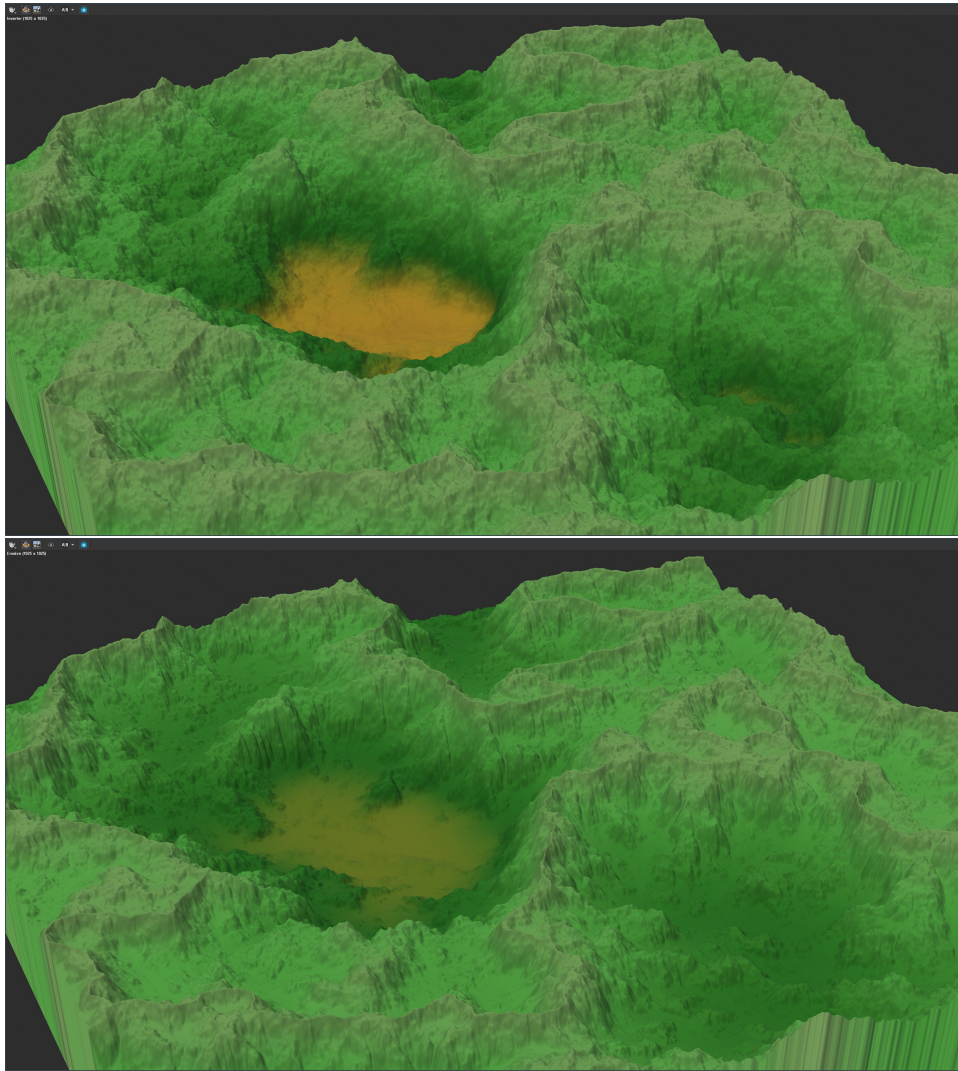


Figure 1.5: Effects of the Erosion node on a ridged Perlin noise terrain in World Machine (above before erosion, below after). Basins are filled with sediment, and many, but not all, disturbances on slopes are smoothed out. Channel erosion can be seen on the mountain on the far side of the deepest basin in the left center.

not riverbeds [18, p. 57]. Also, game developers often use existing satellite data that already contain the influence of hydraulic erosion, and require only the riverbeds to be added to the map.

As such, different methods are used for river generation, which I will describe in Section 1.3. An alternative approach, proposed by this thesis, starts with the user defining the start of the river and the generator creating the rivercourse by following the downward gradient in a terrain created by the previously described methods. The riverbed is then generated along the spline, with the parameters of the river given by following real-world hydrological laws. This section will therefore focus on describing these parameters and laws as they are observed from the hydrology point of view.

As most other methods of river generation, this thesis relies on the Rosgen natural rivers classification [19, p. 169-199]. Rosgen defines nine stream categories - Aa+, A, B, C, D, DA, E, F, G - with each being assigned a set of parameters that defines its cross-section and plan view profile (see Figure 1.6):

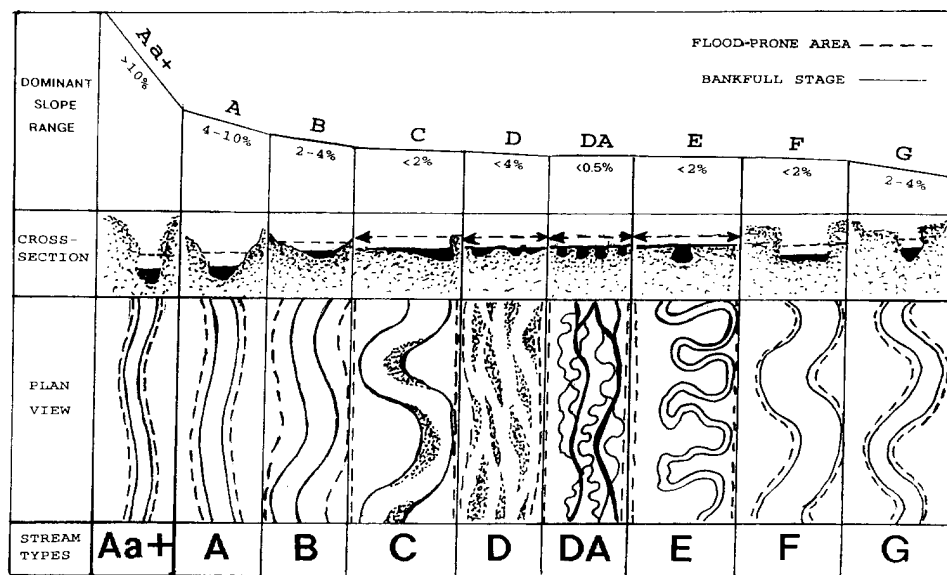


Figure 1.6: Longitudinal, cross-sectional, and plan views of stream types, according to Rosgen [19, p. 174].

- Categories **Aa+** and **A** describe rapidly flowing streams that occur on steeper slopes ($>10\%$ and $4-10\%$, respectively). Those often occur in mountainous regions with bedrock, are narrow, and have sharp riverbanks and V-shaped valleys. The stream often consists of pools, steps, and riffles (see Figure 1.7). Waterfalls, cascades, and rapids can form (see Figure 1.8).
- Categories **B** and **C** describe wider rivers on moderate or low slopes ($2-4\%$ and $<2\%$, respectively). Meandering starts to be present (see Figure 1.9), and the banks are gentler than on Aa+ and A categories rivers. Category B rivers can feature rapids and have narrow valleys. Category C rivers have broad, very gentle valleys, and sediment deposits can be present on the curves' inner side, flattening the river banks even further.
- Categories **D** and **DA** describe braided (D) and anastomosing (DA) rivers that are formed by multiple channels winding over a broad flat valley with little slope ($<4\%$). Depositing sediment forms bars between the channels. Bars on the rivers in the D category comprise glacial debris and large sediment. They erode easily, allowing the channels to change direction often. They often occur when a mountain river transitions into a flatter valley, slowing down significantly and thus depositing carried sediment. Bars on the rivers in the DA category comprise finer soils and are often well-vegetated and stable wetlands (see Figure 1.10).
- Category **E** describes extremely meandering narrow rivers with low banks within broad flat valleys or meadows ($<2\%$ slope). The banks are vegetated and stable, but the surrounding valley is prone to flooding.
- Categories **F** and **G** describe less meandering rivers with high banks protecting the surrounding area from flooding. Both feature high bank erosion rates, with F category rivers being shallow and wide (with slope being $<2\%$) and G category rivers being narrow and deep (with a slightly sharper slope of $2-4\%$).

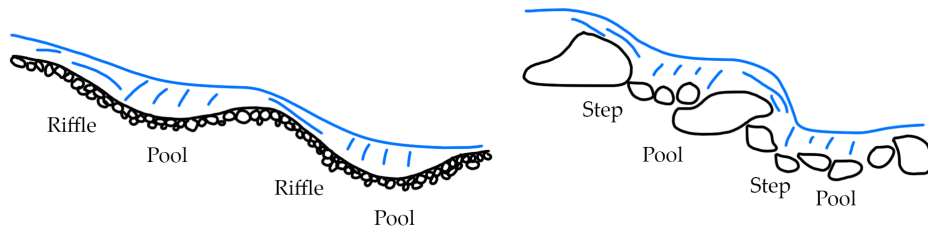


Figure 1.7: Steps, riffles, and pools in categories A, Aa+ streams. Steps are vertical and form when the riverbed comprises larger rocks; riffles are sloped and form in gravel beds. Pools are sections of deeper, slow-flowing water in between those. [20, p. 131]

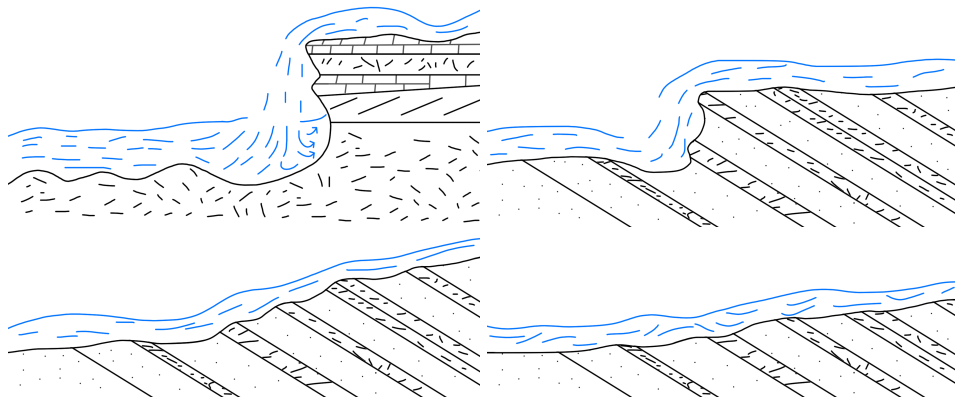


Figure 1.8: In areas with horizontal rock layers, erosion allows the waterfall to recede against the direction of the river by washing out softer layers of soil, causing the harder layers to collapse (top left). In areas with sloped soil layers alternating between a harder rock and softer material, erosion smooths the waterfall out (top right) into cascades (bottom left) and rapids (bottom right). [21, p. 297]

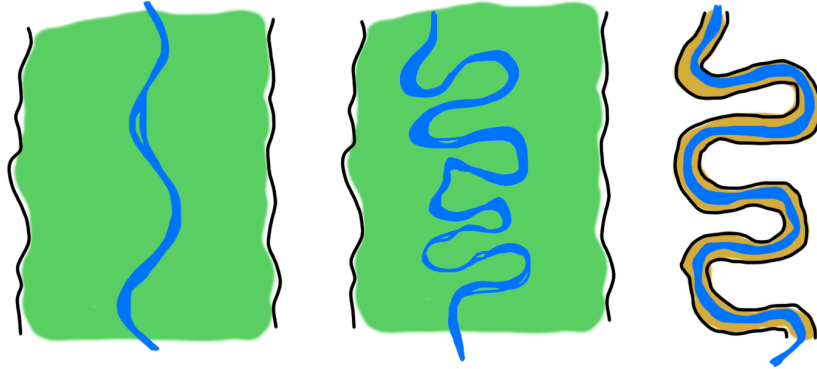


Figure 1.9: From left: Sinoid meanders within a flat valley (typical for category C rivers), tightly-looped meanders within a flat valley (typical for category E streams), and tightly-looped meanders within a confined valley (typical for categories F and G rivers). [22, p. 131]

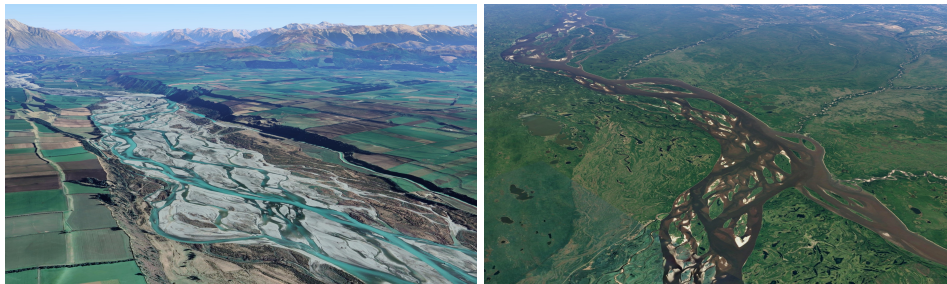


Figure 1.10: Left: braiding Rakaia river in New Zealand running within pronounced riverbanks, with channels separated by unstable gravel bars and blue water due to glacial sediment. Right: anastomosing Lena river in Siberia, with channels separated by semi-stable vegetated floodplain islands and brown water due to mud sediment.^a

^a. Map data: Google, ©2022 CNES/Airbus, Landsat/Copernicus, Maxar Technologies. Left: N -43.63, E 171.80. Right: N 63.31, E 129.82.

Rosgen distinguishes the river categories using the following three ratios:

Entrenchment ratio is the ratio of the bankfull width of the river and the width of the flood-prone belt surrounding the river (see Figure 1.11). The bankfull width of the river is defined as the width of the river water at a bankfull depth. The bankfull depth is the depth at which the water starts overflowing onto the surrounding floodplain. The width of the flood-prone belt is defined as the width of the water surface at double the bankfull depth. The steep banks in narrow mountain streams (Aa+, A) and deeply entrenched streams (F, G) protect the surrounding area from flooding, unlike the low banks of valley streams (in the order of growing entrenchment ratio: B, C, E, D, DA).

Width/depth ratio is the ratio of the bankfull width and average bankfull depth of a stream. Riverbeds of streams on sharper slopes (Aa+, A, B, G) are usually more prone to vertical erosion (the deepening of the river) than to horizontal erosion (the widening of the river). Category E rivers are an exception to this rule, as their banks, stabilized by vegetation, can withstand the river's eroding force. Rivers in the D and DA categories show an extremely large width/depth ratio.

Sinuosity is the ratio of the river length to the valley length. Sinuosity increases with a decrease in gradient and soil particle sizes. As such, the sinuosity ratio ranges from 1.1-1.2 for rivers in the A and Aa+ categories to >1.5 in the E category.

In Section 2.3, I propose a set of parameters used to describe a river's shape and behavior. While these parameters describe the river in a somewhat different approach than the Rosgen classification, they are formed so that they allow the user to describe a river falling under any of the Rosgen classification categories.

One of the objectives of this thesis is also allow the tool generate these parameters. Rosgen's classification categorizes rivers based on their features, which it assumes are related to the slope and the composition of the soil through which they flow. However, more than that is needed to determine the exact river dimensions or channel count.

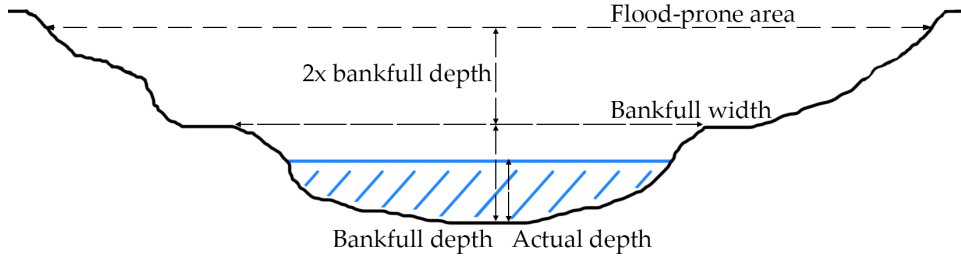


Figure 1.11: A cross-section diagram of bankfull width, bankfull depth, and flood-prone area.

Apart from the slope and soil composition, water discharge is one of the driving variables that affect the river's form. Furthermore, the water discharge significantly impacts the river's dimensions, which is likely the easiest aspect for the player to notice. At the same time, the water discharge is a single variable that can be realistically set either by the environment designer or calculated from the amount of rain over the river's drainage area. As is the case with other topics previously discussed in this thesis, authors in the geology field have proposed precise formulas taking into account variables such as the friction of the riverbed, density of the water, and size of the soil particles. However, most of those values are not available in our case. On the other hand, multiple authors have proposed empirical power formulas for a river's width and depth, using the water discharge as the only input variable.

Based on the fact that water discharge, Q , is a product of the stream depth d , width w , and flow velocity v , Leopold; Maddock [23, p. 8] claim that these three quantities can be approximated with the following expressions:

$$w = aQ^b \quad (1.9)$$

$$d = cQ^f \quad (1.10)$$

$$v = kQ^m \quad (1.11)$$

with a, c, k, b, f, m being constants that can be calculated using regression analysis. Also, since

$$Q = w * d * v \quad (1.12)$$

$$Q = aQ^b * cQ^f * kQ^m \quad (1.13)$$

then

$$b + f + m = 1 \quad (1.14)$$

$$a * c * k = 1 \quad (1.15)$$

Based on a sample of cross-sections of 20 rivers in the midwest United States region, Leopold; Maddock [23, p. 16] propose the respective values 0.5, 0.4, and 0.1 for the exponents b, f, m (to calculate the values in $[ft]$, $[ft]$, $[ft/s]$ from $[ft^3]$). However, according to their findings, values for a, c, k cannot be determined accurately for the whole lengths of the rivers.

Further authors, however, provide those values after an analysis of larger samples of rivers. After analyzing 674 river cross sections across USA and Canada, Allen et al. [24, p. 667] propose the following values:

$$w_{ft} = 1.220 * Q^{0.557} [ft, ft^3/s] \quad (1.16)$$

$$d_{ft} = 0.340 * Q^{0.341} [ft, ft^3/s] \quad (1.17)$$

$$w_m = 2.708 * Q^{0.557} [\text{adj. to } m, m^3/s] \quad (1.18)$$

$$d_m = 0.349 * Q^{0.341} [\text{adj. to } m, m^3/s] \quad (1.19)$$

Finally, Moody; Troutman [25, p. 1261] determine

$$w_m = 7.20 * Q^{0.50} [m, m^3/s] \quad (1.20)$$

$$d_m = 0.27 * Q^{0.39} [m, m^3/s] \quad (1.21)$$

based on a smaller but more variable sample of 226 rivers.

As for the river velocity, while an average velocity could be calculated from Formulas 1.14 and 1.15, this would lead to a simple approximation that grows with flow volume without representing, for instance, a river flowing faster in sections with a steeper slope. In this thesis, I use the river velocity for two purposes: the first is to

roughly simulate the erosion caused by the riverbed (using some of the equations mentioned in Subsection 1.1.3), and the second is to, in the future, pass this value to the water surface material for the visual effects. As such, I have decided to use an adjusted form of an equation proposed by Anh et al. [11, p. 260]:

$$v_{t+1} = (1 - K_s)^{\Delta t_{xz}} * v_t + g * \Delta t_y * K_a, \quad (1.22)$$

where Δt_{xz} , Δt_y denote the XZ distance and Y distance (both in meters) between the points indexed t and $t + 1$, K_s is the sliding constant, and K_a is the acceleration constant.

Since we are interested in adjusting the river speed with distance instead of time, we substitute the time t with the distance d , and calculate the slope S over the distance t instead of the fixed distance.

Other parameters are the meander wavelength and the meander belt width. According to Borgman; Charlton [20, p. 139], the ratio of meander wavelength to the channel width is usually 10 to 14. This seems to be confirmed by Frasson et al. [26, p. 3255], who found the median ratio to be 12.7. However, they note that this method is inaccurate and found that the ratio decreases with increasing channel width (from 21.2 for rivers narrower than 100m to 8.4 for rivers wider than 400m). Regarding the meander belt width, I was not able to find any satisfying relation - Howett [27, p. 104] performed a comparison of multiple previously proposed linear and power models but found that those models were often created based on insufficient sample sizes.

Finally, the channel count for braided and anastomosing rivers depends on the flow volume of the river and the width of the valley to which the river is confined. Crosato; Mosselman [28, p. 4] propose a formula that calculates the expected channel count from flow volume, bankfull width, water depth, slope, dimensionless sediment density, and median grain size. However, since I do not have a way of accurately estimating the last two, I have implemented a custom formula for them instead, which is presented in Section 2.3.

As a thorough description of rivers is not within the thesis' scope nor objectives, this section only described hydrological parameters and laws in a simplified way required for the implementation of the tool. More thorough overviews of rivers, including additional features

of riverbeds and the rules governing their formation are presented by other literature, such as the book *“Fundamentals of Fluvial Geomorphology”* by Borgman; Charlton [20].

1.3 River generation and simulation

This section will overview existing techniques of procedural carving of rivers into terrain based on hydraulic geometry laws. I will also overview techniques used by terrain generation software and in the game development industry, first for generating entire river networks and then for generating rivers one at a time.

While this chapter describes all important methods of river generation, more thorough overviews exist, such as the article *“How to build a digital river”* by Brown; Pasternack [29].

1.3.1 Generating river networks

The most intuitive method to generate a river network over an entire heightmap is to compute the drainage area adherent to each point. Empirical methods used in the previous section can then be easily used to generate the geometry of the river based on variables such as the flow volume, slope, shape of the surrounding terrain, and estimation of the amount of sediment.

A somewhat inverted method was first proposed by Kelley et al. [30, p. 263-268]. The algorithm takes an outline of an island landmass as an input, a map of slope inclines of the future valleys, a line defining a rough representation of the main river, and a mean distance between tributaries. Tributaries are randomly generated, starting at the river’s end and continuing against the river flow. The angle at which tributaries join the river is then computed based on the desired slopes incline, and the length of the tributary is set to a random value within a user-set range. The process is then recursively repeated for the tributary (while decreasing the length of each subsequent tributary) and continued on the main river. The result is a fractal network of randomly generated streams. The final terrain is then generated by creating an optimized surface using the minimal possible amount

of polygons while simultaneously limiting the displacement of the generated river network and respecting the defined slope incline.

While the terrains generated by this method look realistic, the user has minimal control over the result. Therefore, recent authors focus on presenting techniques that allow the user to shape the desired terrain layout more precisely. G  nevaux et al. [31] present a method that, similarly, first generates a river network, and only then the surrounding terrain. However, in addition to the terrain outline, it allows the user to provide terrain and river slope maps (where a higher value refers to a sharper incline, not a higher elevation). Empirical methods are used to generate river mouths, and the network is expanded by adding new nodes further inland or branching the river. The elevations of the points are created based on the river slope map. Once no further points can be added, a rough heightmap is created by multiplying the distance from the nearest river by the value in the terrain slope map. Each segment of the river network is then assigned a Rosgen river category based on its slope and proximity to the coast - meanders are created by generating random tangents on the curves. The probability of branching increases the closer the river is to the coast. Each river category has a curve assigned defining the shape of its cross-section, and the curves are interpolated to ensure continuity. Finally, additional work is done to add more detail to the terrain, such as adding more detailed hills and mountains.

Overall, the method produces large terrains in a short time but still does not provide the user the ability to specify the paths of rivers as well as their parameters. Therefore, the authors adjusted the process in [32], where the user defines the terrain by placing feature primitives, such as mountains, hills, and lakes, by drawing shapes on the terrain. The primitives are then blended together. Rivers can be defined by a curve, where nodes of the curve are assigned different cross-sections (represented by 1D piecewise quadratic functions). For the segments between those nodes, the cross-sections are blended accordingly.

In addition to the carving of the riverbed, Peytavie et al. [33] add the possibility of also generating the river surface. The river network is generated over a discrete grid of cells, with rivers constantly flowing down to the cells with the most prominent elevation decline. An empirical method is used to calculate the flow volume at each cell, as well as the drainage area and the river slope. Rivers are formed by



Figure 1.12: A rendering of a generated river by Peytavie et al. [33].

following cells with a flow volume higher than a specific user-defined threshold. Parts of the rivers are then assigned their Rosgen categories, and the curve defining the river is refined. Meanders can be added at this point if the slope of the river is low and the flow is moderate. Steps, pools, cascades, and waterfalls can also be added by modifying the curve's Y offsets. Again, cross-section curves based on the Rosgen classification are instantiated along the rivers and blended. Finally, river flow primitives are seeded along the river to create an animated river surface. Thus, the river can be either calm or turbulent and contain waves, cascades, vortexes, and ripples. For instance, cascades followed by waves are generated at steps and riffles on category A rivers, and turbulences are generated on straight segments of category C rivers to emulate rapids. The final water surface is blended using a method similar to the one proposed for blending terrain primitives by [32].

While the last two methods require the user to prepare cross-section definitions manually, they also give them significant control over the process by allowing them to create rivers one by one and adjust their shape in a precise manner, making them suitable even for game development environment design. In the following subsection,

I will focus on methods of creating these cross-sections from a set of user-friendly variables.

1.3.2 Generating rivers by modelling riverbeds

This subsection focuses on methods proposing the most suitable approach for video games, as they give the designers almost total control over the shape of the river. At the same time, all of the time-consuming sculpting, including the generation of the cross sections, is done automatically. This allows the designers to iterate over variable terrains quickly.

Compared to the previously mentioned ones, this approach to generating riverbeds is relatively young. The first such method is described by Brown et al. [17]. The method generates a heightmap representation of the river based on the average bankfull width, depth, slope, median sediment size, critical Shield stress (force at which erosion starts), valley heights, and valley widths. Taking into account the empirical laws governing hydraulic geometry and the oscillatory nature of many of the river's parameters, Brown et al. propose a model capable of generating short segments of rivers without requiring the user to model cross-section-defining curves directly. However, the method was presented on short segments of rivers and only supports streams with a single channel.

Since then, similar methods have, however, seen use in game development. A tool based on similar principles, used in the development of game *Horizon Zero Dawn* is showcased in [34]. The tool was designed as a plugin for SideFX Houdini¹⁰. First, the user draws a curve on the terrain to represent a rough outline of the rivercourse. Then, over multiple iterations, the tool slides the points of the curve down the terrain gradient to push it into valleys already present on the terrain. To prevent the river from flowing upwards, the tool then limits the elevation of every curve point to equal or lower than the point before it, forcing the river to cut canyons through the higher terrain. The user can specify the width of the river and the valley, river depth, bankfull depth, and valley height. An irregular $7 \times n$ grid mesh is then created

10. A modeling/animation program commonly used for procedural generation in video game development. Houdini allows the user to create plugins using python or visual scripting. <https://www.sidefx.com/>

along the curve, with n being the length of the curve in segments. The center column forms the riverbed (at the elevation set by the curve lowered by the user-set channel depth), with the following columns forming banks (transition to valley), valley (at the elevation of the curve), and the edge columns forming valley walls (transition to the original terrain). By offsetting the channel's position within the width of the valley, sinoid meanders can be added. The final terrain is then created by blending the river mesh and the original terrain. Since multiple faces of the valley mesh can overlap at sharp curves, the one with the lowest elevation is always chosen as the result elevation.

Finally, Tokarev; Trümpler [35] describe expanding this method to generate the water surface mesh, including flow maps, masks for foam, ripple textures, and positions for particle effects. The result data are then exported into Unreal Engine using the Houdini Engine plugin for Unreal, which also allows the user to make changes to the river directly in Unreal, with Houdini automatically regenerating all the wanted mesh and texture data.

From the terrain generation software, only World Machine provides a river generator tool as of the writing of this thesis¹¹. The user can create a whole river network automatically (with the layout of rivers being determined by the water flow volume) or by manually drawing curves on the terrain. In that case, the river's elevation is based on the elevation of the existing terrains, and canyons form wherever the river would be forced to flow uphill, similarly to the method mentioned above. The user can set the bankfull width and depth of the river, select a basic channel shape, and set the valley dimensions and shape. Sinoid meanders can be added by offsetting the river along the flow vector normal. The tool is currently labeled as experimental and contains some irregularities. The valley shape in sharp corners often creates sharp edges and other artifacts. Meanders can only be created on the river, not the valley, making creating realistic-looking narrow canyons impossible.

11. According to the World Creator website, a Procedural River Generator tool is planned in a future update, <https://www.world-creator.com/features.html>. Also, Gaea does feature a tool that generates entire river networks. However, it is extremely limited in that it does not allow the user to influence the rivers themselves in any meaningful way, apart from setting a global bankfull width and depth multipliers for the entire terrain at once.

A common characteristic of the techniques mentioned in this subsection is the speed at which the terrain changes can be performed. Since the shape of the riverbed cross section is calculated using simple interpolation between multiple values based on the distance from the generating curve, they are much faster than the evolutionary methods simulating erosion. As such, it is possible to often achieve an almost real-time reaction to changes in the shape of the spline or the river parameters, despite performing the algorithm on the CPU (as is the case with the method implemented in Houdini).

On the other hand, the main drawback of those methods is the necessity for the user to still set the dimensions of the river manually. Also, the method of sliding the spline down the hill seems to be problematic - the points are also moved if the gradient and the curve tangent are similar, which results in segments of the curve that are already perpendicular to the terrain contour simply sliding down the mountain, eventually moving the river from mountainous areas to the flatlands. As such, this thesis proposes a new method of creating the optimal riverbed and employs the relations mentioned in Section 1.2 to generate the river parameters automatically while allowing the user to change them if desired. As for the carving of the river into the terrain, the proposed method will also follow a novel approach respecting to the technical specifics of the game engine in which it was implemented.

2 Proposed method

Based on the objectives described in the introduction, I have decided to split the river generation into three steps:

1. generating a spline representing the rivercourse based on the terrain gradients,
2. generating parameters, such as the river dimensions, valley shape and other features along the course of the river,
3. modifying the terrain based on the results obtained from the previous two steps.

One of the main requests made by the consulted environment designers was that while all three steps can be used together to create a river from scratch, it should also be possible for the user to supply their own data, and the tool should be able to work with them as expected. For instance, the designer might want to import a spline defining the course of a river from an actual map, as these data are often easily accessible on the internet. Also, while the proposed method can propose parameters that make the generated river look realistic, a designer might want to change these for gameplay purposes, such as making the river shallower and wider at points to allow the player to wade through the stream on foot, or for visual purposes, such as making the river braided in wetlands biomes to emphasize the location's atmosphere. Dividing the process into three steps is therefore suitable not only from the technical standpoint but also for fulfilling the objectives of the thesis.

Over this chapter, I will present the three steps forming the entire process independently.

2.1 River representation

Since the method, unlike, for instance, erosion-based terrain techniques, focuses solely on the generation of rivers, we first need to define how the river itself will be represented in the game engine.

As defined by the thesis assignment, the reference implementation was done in Bohemia Interactive's proprietary Enfusion Engine.

While I have tried to design the process in a way that should be suitable for any modern 3D engine, which supports simple spline objects and heightmap-based terrains, some aspects of the method had to be adapted to the specifics of the engine in question. This is most clear in the last step of modifying the terrain, which is tightly tied to how the engine represents the terrain itself and what API it exposes to the developer to perform changes to it. However, it also had a negligible influence on other parts of the process.

I have used spline curves to represent the rivercourse itself. Thanks to their ease of use both in terms of implementation and manipulation by the user, splines are commonly implemented in most game engines available on the market. In the Enfusion Engine specifically, Catmull-Rom spline interpolation is used. Spline entities (an object managing a spline describing a single shape) can be created and modified using a *Vector Entity tool*, which is part of the engine's World Editor. Thanks to the splines' inherent properties such as the precision over the control points, fast evaluation, and tangent continuity, they are suitable for representing any continuous entity, be it not only rivers but also roads, railroads, or rough definitions of terrain features such as canyons. When carving the river, we evaluate the spline at small intervals, eventually producing a polyline. A similar method is already used in the Enfusion Engine World Editor to create water surfaces for rivers - a mesh grid is created along the curve, and a material using mesh space UV for texturing is applied. Therefore, a single spline entity can be used to generate both the riverbed and the water surface. Other generators, such as the road generator or the wall/fence generator, use a similar method by either projecting material onto the terrain or spawning objects along the curve.

Furthermore, Enfusion allows the user to add `PointData` objects to spline control points. These allow the user to modify the behavior of generators on a per-segment basis, such as changing the width of the water surface mesh created by the river surface generator. Instances of classes inheriting from `PointData` (`RiverGeneratorPointData`, in this case) can contain various variables, ranging from booleans, through numeric values, to vectors, strings, and arrays. The World Editor will automatically allow users to edit them in the engine using the *Vector Entity tool*. This introduces a minor limitation to my approach in that the user can only define river parameters per control point

- if a change in river parameters is needed between existing control points, a new point has to be added. On the other side, not all control points need to have river parameters assigned - if a point does not have any, the tool finds the nearest previous and subsequent control points that do and interpolates the parameters accordingly. An exception to this are the spline's first and last point, which are assigned a `RiverGeneratorPointData` object with default values if the user does not assign any.

While some additional structures are created during the various steps of the process, they are only temporary and, as such, not important for the design of the method itself. Therefore, I will address them in Chapter 3.

2.2 The optimal rivercourse

In the methods described in Subsection 1.3.2, the user would draw a rough representation of a rivercourse, which would then be procedurally adjusted to follow existing valleys. On one side, this method gives the user slightly more control while still allowing them to have the tool refine the rivercourse automatically. At the same time, however, they can also easily create rivers that carve deeply into hills or sides of the valleys, even if a much more viable path is available. This allows the river generator to influence unrealistically large changes to the terrain by introducing canyons into existing plateaus if the user draws an unsuitable line.

To further automatize the process and decrease the chance of a user-induced error, I have decided to examine a different approach. The presented method creates the river by taking only the terrain heightmap and the position of the river source as input. Alternatively, if the spline representing the rivercourse already contains some points, the user can either decide to discard the points and start the process from the beginning or to keep them and only add new points after them. Because of this, if the user is unsatisfied with a specific part of the spline, they can easily adjust it manually using the Vector Entity tool and restart the rivercourse generation for the remaining segments.

The user can also set the additional parameters:

Step size defines the expected distance between consecutive spline control points. With increasing step size, the distance between the consecutive points increases, which makes the river direction less volatile. A larger step size also lowers the probability that the process will end prematurely due to getting stuck in a local minimum.

Target elevation is the elevation in world space, at which the river generation will end, such as an elevation of the water surface of a sea or a lake, to which the river should enter.

Maximum generated points count limits the maximum number of points that can be generated at once. Users can generate the rivercourse in shorter segments if they want more control over the process.

New points are then added one by one based on the geometry of the terrain using the following algorithm:

1. Estimate the next heading of the river by protracting the last segment: $\overline{pt_{n+1}} = pt_n + d_s * t\hat{g}_n$, where pt are points of the spline, n is the index of the last existing point, d_s is the step size and $t\hat{g}_n$ represents the normalized vector of $\overrightarrow{pt_n, pt_{n-1}}$.¹ If only one point exists, then $\overline{pt_{n+1}} = pt_n$.
2. Limit the set of possible pt_{n+1} - if multiple points already exist, create two circles:

$$O_1 : c = pt_n, r = \frac{4}{3}d_s, \quad (2.1)$$

$$O_2 : c = \overline{pt_{n+1}}, r = \frac{2}{3}d_s, \quad (2.2)$$

and connect them at their furthest points from the line $\overrightarrow{pt_n, pt_{n+1}}$. Restrain the next added point to be inside this shape. If only one point exists, the next point has to be in a $4d_s$ radius around pt_n .

1. One might argue that a more precise tangent estimate $t\hat{g}_n$ as the tangent of the river at the last point estimated so that the angle of a polyline connecting points pt has the same angle at pt_n as it does at pt_{n-1} (since we are not yet able to exactly determine the tangents according to Catmull-Rom, which for pt_n require us to know positions of pt_{n-1}, pt_{n+1}). However, this would cause the river to keep turning infinitely after any curve due to a feedback loop in the tangent definition.

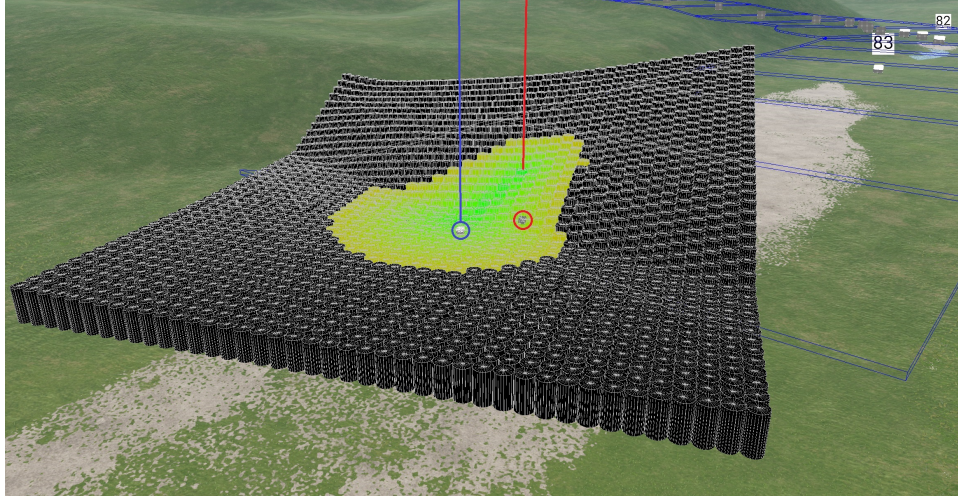


Figure 2.1: Shape restricting the candidates for the next river point, with the final score of the points visualized by the elevation of the cylinders. River is coming from the top right corner of the image, with the previous added point circled red (located under the red vertical line), and the newly added point circled blue. Points represented by black cylinders are not considered for the next river point.

3. Retrieve the set of all candidate points pt_c of heightmap that satisfy the previous condition.
4. Skip all candidate points for which $pt_{c|y} - K_{ht} > \min_{p \in pt} p_y$, where $pt_{c|y}$ and p_y denote the y components of points pt_c and p , and K_{ht} is a height-tolerance constant.
5. For each candidate point, compute the **direction score** s_r for each point:

- If the spline only contains the first point, then

$$s_r = \frac{|\overrightarrow{pt_c, pt_n}|}{2d_s}. \quad (2.3)$$

- If multiple points already exist, then we first create pt_c 's projection p_{ab} onto the line $\overrightarrow{pt_n, pt_{n+1}}$ and define t_{ab} as the

position of p_{ab} along the line normalized to $[0, 1]$. Set

$$s_r = \frac{|\overrightarrow{pt_c, pt_{ab}}|}{lerp(\frac{d_s}{3}, 2\frac{d_s}{3}, clamp(t_{ab}, 0, 1))}, \quad (2.4)$$

where

$$lerp(a, b, t) = a + t * (b - a) \quad (2.5)$$

$$clamp(x, a, b) = max(a, min(x, b)). \quad (2.6)$$

6. For each candidate point, compute the **distance score** s_s for each point:

$$c_{exp} = clamp(d_s - |\overrightarrow{pt_c, pt_n}|, 0, 1) \quad (2.7)$$

$$s_s = max(1 - t_{ab}, 0.1 * 10^{c_{exp}}). \quad (2.8)$$

7. Pick the best candidate point based on the lowest **final score** s :

$$s = pt_{c|y} + K_r * s_r + K_s * s_s, \quad (2.9)$$

where K_r and K_s are constants. Add the best candidate point to pt .

The tool uses $K_{ht} = 3$, $K_r = 1$, and $K_s = 5$. These values have been heuristically determined after tests on multiple various terrains. It should be noted that since $s_r \in [0, 1]$ and $s_s \in [0.1, 1]$, K_r and K_s can be thought of as a maximum "height" that can be added to the points based on their XZ position to deter the algorithm from creating rivers that often change directions, or to prevent it from creating unnecessary amounts of points where it is not needed. The new point is finally added at the associated XZ position, and the point's Y position is set to the terrain elevation at that height. The process is then repeated.

Finally, the spline generation can stop for one of four reasons:

- the river has reached a target elevation set by the user (for instance, the sea level),
- a maximum count of generated points specified by the user has been reached,

- a local minimum in the terrain has been reached, and the river cannot continue further without creating a canyon or
- the river has created a cycle (which can happen on flat sections of the terrains with almost zero terrain gradient).

After the spline is created, the final step is to optimize it by removing any unnecessary points. These can be points located on a long straight river segment or too close to other points. This can also smoothen the river, as places with a high concentration of points will often lead to a river that changes direction erratically. To control the extent of this process, the user can set the **Minimum point deviation**. To remove points on the straight segments, we remove every point whose distance from a line between its two neighbors is less than the minimum point deviation. To smoothen the spline, we remove all points whose distance to either of its neighbors is less than this value and the angle between the lines to them is less than 90° . Points are iterated from the beginning and removed during the iteration process. Despite the approach being very straightforward and making only greedy choices, it removes the points considered unnecessary by our metrics successfully enough.

2.3 River cross section parameters

As the output of the first step is only a spline describing the rivercourse, the next step consists of generating the parameters describing the cross-sections of the river. For that, we use the following parameters:

River width is the distance between the edges of the water-covered river perpendicular to the river tangent.

River depth is the elevation difference between the water surface and the river's deepest point (known as the thalweg).

Riverbed cross-section shape defines the thalweg position within the river cross-section. Especially in category C rivers, which are fast-flowing and have noticeable meanders, the thalweg does not follow the center of the river but, at curves, approaches the outer riverbank. The user can model the cross-section using a

curve tool, which gives them further control over the shape. The curve is defined in a $[-1, 0]$ to $[1, 1]$ rectangle, where $x = -1$ denotes the transition to the left bank, $x = 0$ the spline position, and $x = 1$ the transition to the right bank.

Riverbank cross-section shape lets the user model the cross-section of the banks using a curve tool similar to the riverbed shape. As such, they can create wide rectangular riverbanks (typical for the category D streams), linear V-shaped banks (typical for categories Aa+ and A streams), or parabolic U-shaped banks (typical for categories B and G streams). The curve is defined in a $[0, 0]$ to $[1, 1]$ rectangle, where $x = 0$ is the transition to the riverbed, and $x = 1$ is the outer edge of the influenced terrain.

Riverbank height is the vertical extent of the riverbanks defined by the riverbank shape (between the original terrain and the bankfull river surface elevation). In the presented approach, the user can use the riverbank to represent both the banks and the nearest floodplains of the river.

Left, right riverbank distance define the horizontal extents of the riverbanks defined by the riverbank shape. It is possible to define the left and right riverbank distance separately to prevent the river from aggressively cutting into the hills if it is flowing on the edge of the valley.

Riverbank, riverbed height noise are the heights of a layer of 2D Perlin noise added to the generated terrain to add further variety.

Channel count is the number of channels that form the river. To allow blending between different channel counts, we allow the user to enter real numbers, where the fractional part acts as a blending parameter for the last channel's position. For instance, channel count 5.5 results in 5 independent channels, plus an additional channel halfway merged into the fifth channel. If multiple channels are formed, they will randomly oscillate within the entire span of the river's banks.

River width and depth, bank height and distance randomness define the maximum height of 1D Perlin noise added to each river dimension.

Bank, river meander length are the wavelength of meanders, i.e., the valley distance, after which the meander completes both the left and right turns.

Bank, river meander width ratio define the amplitude of meanders of both bank and the river, expressed as the ratio to the meander distance.

River meander tightness blends between sinoid and tightly-looped meanders.

River meander randomness defines the strength of meanders distortion. Meanders are distorted by warping the space in which their coordinates are computed. This applies only to tightly-looped meanders.

These parameters are applied at each point and can be easily interpolated between the points. While the user can set these parameters separately for each control point, it could be time demanding for larger rivers. For this reason, assigning parameters to points is optional, and the tool interpolates the data from the nearest previous and next points that do contain the parameters based on the spline distance between them. In this case, the point only keeps the indices of the following and previous suitable points and the linear interpolation alpha parameter. These are computed before terrain generation and then discarded. Therefore, these data are not saved to the hard drive with the rest of the map but are still accessible during the terrain generation process.

Generating the parameter values is, again, done in order from the river source to the river's mouth. The user can also limit the parameter generation to only a specific section of the river (specified by the indices of affected control points). The algorithm requires only two inputs - one describing the water discharge volume, and the other the probability of the formation of multiple channels. Both are expressed as curves over the actual length of the river (or the segment for which parameters are being generated).

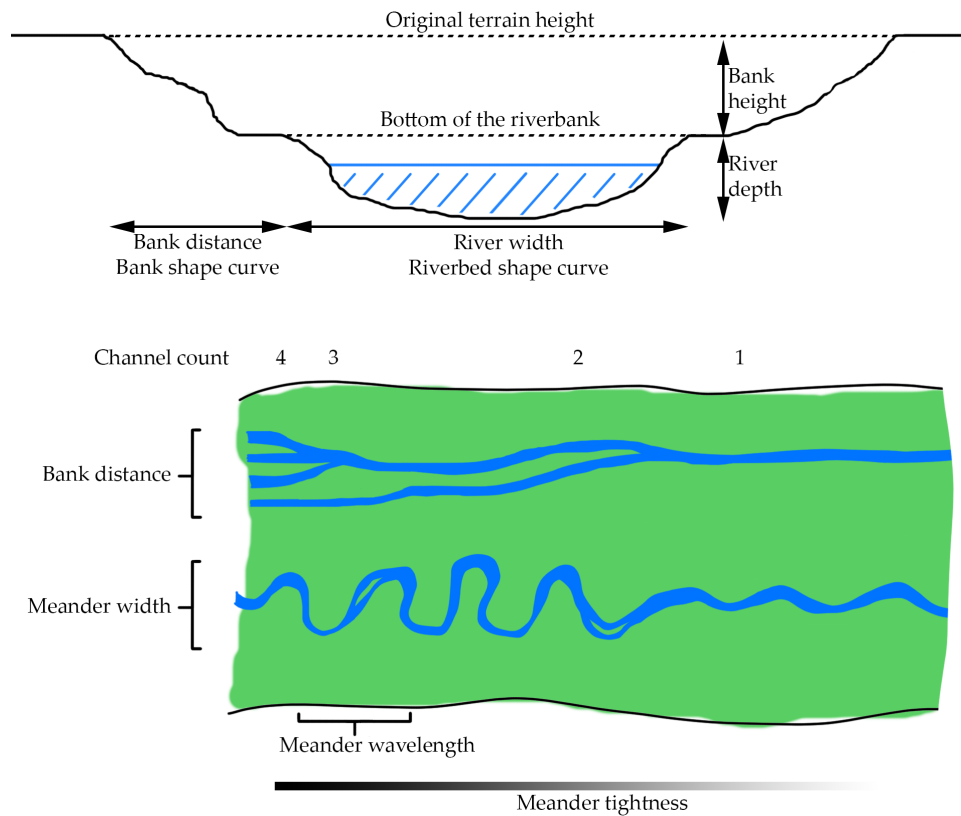


Figure 2.2: Parameters describing the shape of the river. Cross section parameters describing the dimensions of the river on top. Planar parameters describing behavior of braiding and meandering rivers on bottom.

I have already described most of the relations needed to estimate all those parameters in Section 1.2. We will need the water flow volume, the river's slope, and approximations of the river's curvature, speed, and the amount of carried sediment to compute them. To determine some values, we will use some of the ratios described by the Rosgen classification for various rivers, such as the relation between the slope and the entrenchment of the river.

The algorithm then goes as follows:

1. We tessellate the spline using the Catmull-Rom interpolation, and define cp as the sequence of i control points, tp as the sequence of j tessellated points. We define a function $tcp(tp)$ which, for any tessellated point, returns the index of its preceding control point. If a tessellated point is the same as a control point, the index of that control point is returned.
2. We determine the following sequences, each with j members:
 - (a) d as a sequence of horizontal distances between the tessellated points (where d_t is the length of $\overrightarrow{tp_t, tp_{t-1}}$ projected on the XZ plane).
 - (b) e as a sequence of elevation differences between the tessellated points (defined as $e_t = tp_{t|y} - tp_{t-1|y}$, where the $|_y$ suffix denotes the y component of the vertex, and the value is negative if the river flows downhill).
 - (c) q as a sequence of the volume of water flows at tessellated points, which we evaluate from the flow volume curve, with input to the curve being the distance of the tessellated point from the river start divided by the river length. We can then calculate the estimated river width at each point using the Equation 1.20.
 - (d) g as a sequence of slopes (the ratio between the distance and elevation change). For more stable results, we do not evaluate the slope only against the previous tessellated point - instead, we take the last point, whose spline distance from the current point is at least $2 * w_r$, where w_r is the current width of the river.

- (e) v as a sequence of river speeds at tessellated points using the Equation 1.22:

$$v_{t+1} = (1 - K_s)^{\Delta t_{xz}} * v_t + g * \Delta t_y * K_a.$$

The tool currently uses $K_s = 0.025$, $K_a = 0.2$, as these values led to the most realistic results on the tested terrains.

3. Using the sequences computed in the previous step, we can now evaluate for each control point cp_m (with final parameters **highlighted**):

- (a) **River width** as $w_r = 7.20 * w^{0.50}$ from Eq. 1.20.
- (b) **River depth** as $d_r = 0.27 * w^{0.39}$ from Eq. 1.21.
- (c) Point curve coefficient $c_c = \frac{w_r * \angle_m}{\pi * avg(d_m, d_{m+1})}$, where $\angle_m$ denotes the angle of the polyline at the currently processed point cp_m .
- (d) River slope coefficient $c_s = clamp(10 * g_m, 0, 1)$.
- (e) **Riverbed cross-section shape** as an U shape defined by five points: two anchors at $[0,1]$ and $[1,1]$, center at $[0.5, 0]$, and two points defining the slope at $[0.1, 0.2]$ and $[0.9, 0.2]$. We then shift the center point using the curve coefficient by $clamp(c_c, -0.3, 0.3)$, and shift the slope-defining points left or right accordingly. Then, we also move the slope-defining points up by $0.5 * c_s$ to create a V shape on 10% and higher slopes (to imitate the behaviour of A and Aa+ Rosgen category rivers, see Section 1.2).
- (f) **Riverbank cross-section shape** by blending two curves consisting of four points: an S-shape curve ($[0,0]$, $[0.2, 0]$, $[0.8, 1]$, $[1,1]$) and a linear curve ($[0,0]$, $[0.2, 0.2]$, $[0.8, 0.8]$, $[1,1]$), again using c_s as the blending alpha parameter.
- (g) Amount of carried sediment s . The tool uses the method originally proposed by Musgrave et al. [9, p. 45] adjusted

to use distance instead of time as the function input:

$$s_{cap|m} = K_c * q_m * v_m, \quad (2.10)$$

$$\Delta s_m = \begin{cases} (1 - (1 - K_d)^{d_m}) * (s_{m-1} - s_{cap|m}) & \text{for dissolution if } s_{m-1} > s_{cap|m} \\ (1 - (1 - K_s)^{d_m}) * (s_{m-1} - s_{cap|m}) & \text{otherwise for sedimentation,} \end{cases} \quad (2.11)$$

$$s_m = s_{m-1} - \Delta s_m. \quad (2.12)$$

The implementation uses $K_c = 250$ proposed by [11, p. 260], however, due to the changes to the equation, it was necessary to adjust for $K_d = 0.0015$ and $K_s = 0.0012$.

- (h) Local probability of forming multiple channels, evaluated from the user supplied curve using the river distance divided by river length as the input. We then calculate the braiding threshold t_{br} based on this probability and the slope of the river (with higher slope lowering the chances of braiding).
- (i) Maximum riverbank width, $bw_{max} = (1 - c_s) * w_r$ for each side of the river. I have decided to limit the width of the banks of single-channel rivers generated by the tool to w_r on each side, since larger terrain features are expected to be already present on the heightmap. However, if the braiding threshold t_{br} allows the formation of multiple channels, we raise the total width of the banks, including the width covered by water, to up to $4 * w_r$.
- (j) **Riverbank left/right distance** by doing a ray cast against the terrain. The tool uses $cp_m + [0, d_r, 0]$ as the start of the rays, the right and left spline bitangents at cp_m as the direction, and bw_{max} as its length on each side. Bank distances on each side are then set to the distance travelled by the associated ray before hitting the terrain (or to bw_{max} if the ray failed to hit the terrain).
- (k) **Riverbank height** to $\frac{d_r}{5}$ for points with $<1\%$ slope, up to d_r for points with $>5\%$ slope.

- (l) **Channel count** from the width of the river (wider river can support more channels), width of the banks (more channels fit into wider banks) and the sediment change (higher sediment deposition results in river being divided by more bars):

$$ch_{max} = clamp(w_r/8, 1, 10) \quad (2.13)$$

$$ratio_{r/b} = \frac{bw_{left} + bw_{right}}{w_r} \quad (2.14)$$

$$ratio_s = clamp(0.05 * \Delta s_m, 0, 1) \quad (2.15)$$

$$ch = clamp(3 * ratio_{r/b} * ratio_s, 1, ch_{max}) \quad (2.16)$$

The tool also limits the change between two neighboring control points based on their distance to prevent fast fluctuations in the channel count over short segments.

- (m) **Randomness ranges** are simply set to 10% of the associated dimension.

4. Finally, we create the `RiverGeneratorPointData` structures holding the parameters and assign them to the points.

As for the meander parameters, while a sufficient model does exist for estimation of some of them (see page 30), their automatic generation is not yet included in the tool due to issues with the meanders implementation. These issues are discussed in Subsection 4.1.3.

I have also decided not to automatically create terrain height noise, as its function is purely visual. While it could be derived, for instance, from the flow speed (a faster flow will erode rough riverbeds easier), the user might find it undesirable in many cases.

The user can, again, decide to make changes to the river by moving the points or changing the values of the generated parameters as they see fit. After this step, it is possible to use all the gathered information to carve the river into the terrain.

2.4 Evaluating the terrain changes

One of the requests by the Bohemia Interactive environment design team was that the tool should be easily expandable and parts of the

tool should be repurposable for generating not only rivers but also other terrain features (see page 5). For this reason, core parts of the thesis implementation are programmed inside the game engine in the C++ language, while the functionality related to rivers is programmed in the Enfusion Engine's script, Enforce. Users can create custom generator entities in the script, such as road and rail embankments, mountain ranges, lakes, or other geographic features whose shape can be roughly described with a spline.

Since the two previous steps, generating the spline and generating the river parameters, are related solely to river generation, and engine-level features required for their implementation are already present in the engine, I have implemented them entirely in the script language. However, due to heavy reliance on the low-level terrain code and performance constraints, internal parts of terrain generation had to be implemented in the game engine itself. This leads to splitting the functionality required for changing the terrain into two parts:

SplineTerrainGeneratorEntity class representing the generator in the engine code. It is implemented in C++ and associates heightmap points with spline control points that should influence them while trying to minimize the number of required script calls to actually generate the new elevations (these two operations are the most time-demanding in the whole process). It then prepares other data useful for terrain generation, such as the distances between the points or the distance of heightmap points from the spline segments. These are then passed to the script class, which evaluates and returns the actual changes to the terrain, which are then written into the new heightmap.

RiverGeneratorEntity class representing the river generator in the script. It extends the **SplineTerrainGeneratorEntity** class by implementing the `GenerateTerrain()` function, which evaluates the new elevation of the heightmap point. However, it also adds other features related to rivers, such as the implementation of the generation of the spline and parameters).

Due to the approach of modifying terrain implemented by the Enfusion Engine's terrain API, terrain generation, unlike the two previous steps, cannot be done progressively from the source of the river

to its mouth. Instead, when we request a part of the terrain to be generated, the engine divides the area into multiple smaller rectangular patches, which are then modified and applied to the terrain heightmap separately. This is further described in Section 3.1. This means that the terrain generation is done separately for each heightmap pixel in an order the user cannot influence or reliably predict. The only information given to the `SplineTerrainGeneratorEntity` when a terrain regeneration is requested is a rectangle of the heightmap to be regenerated.

`SplineTerrainGeneratorEntity` then processes this into the following set of values provided to `GenerateTerrain()` as arguments:

- Sequences of the spline control points cp , tessellated points tp , and a function $tcp(tp)$ which, for a given tessellated point index, returns the index of the last preceding control point.
- A sequence of distances of every tessellated point from the beginning of the river x (not from the previous point, as during the parameter generation).
- The position of the generated point in world coordinates, p_{ws} , where the y component contains the elevation including any previously generated rivers.
- The position of the generated point in terrain heightmap coordinates, p_{ts} .
- The elevation of the terrain at the generated point before any river has been generated, h_o . Thanks to this value, multiple overlapping rivers do not create an exceedingly deeper riverbed, but only the effect of the deepest river is kept.
- The point on the spline closest to the generated point p_{sp} , as well as its position on the spline defined by the indices of the previous and next tessellated spline points $i_{t|p}, i_{t|n}$, and the distance x_{tab} along the line between the two normalized to $[0,1]$.
- The signed distance from the spline y - in this case, a negative value means that the point is on the left side of the spline (considering the direction of the spline to be from the control point

with index 0 to the control point with the highest index) and a positive value means that the point is on the right side.

- The inverse of resolution of the terrain heightmap.

In this section, I will only describe how we evaluate terrain changes from these values. How they are computed in the game engine code by `SplineTerrainGeneratorEntity` will be described in Section 3.2.

First, we retrieve the river parameters applied to the particular point along the river. We use the position on the spline described by $i_{t|p}$, $i_{t|n}$ and x_{tab} and the function $tcp(tp)$ to get the indices of the previous and next control points $i_{c|p}$, $i_{c|n}$ and the distance along the spline between those points x_{cab} . As discussed on page 45, each control point that is not assigned a set of parameters instead keeps indices of previous and next control points that have those. This means we can evaluate parameters at each control point. To get parameters for a specific point along the river, we only need to interpolate the values between the two control points using x_{cab} as the blending parameter. However, a simple linear interpolation would lead to abrupt changes in the first derivative of these functions at control points. Instead, we use S-spline interpolation with the polynomial $6x^5 - 15x^4 + 10x^3$, which leads to a more visually pleasing result.

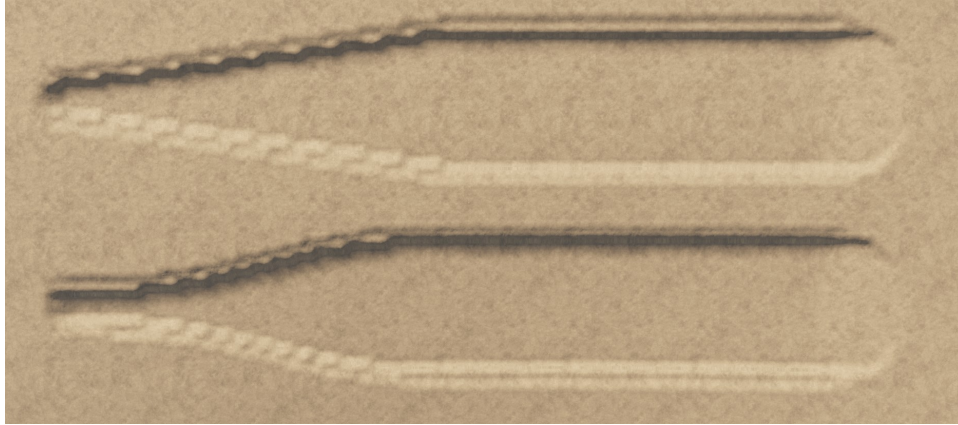


Figure 2.3: A comparison of linear (top) and S-spline (bottom) interpolation of parameters between control points. On larger rivers, S-spline interpolation produces more realistic results.

The process of carving a river without any braiding or meanders is relatively simple. The effect of the river generator is limited to the points whose distance from the spline y is less than $\frac{w_r}{2} + w_{bk}$, where w_r represents the river width and w_{bk} the bank distance on the particular side of the river. If $|y| < \frac{w_r}{2}$, then the point we are currently generating lies in the riverbed, and if $0 \leq |y| - \frac{w_r}{2} < w_{bk}$, it lies on the riverbank. By sampling the riverbed or riverbank cross-section curves, we get an alpha parameter α , which represents the blending between the full depth at $\alpha = 1$ and no effect on the terrain at $\alpha = 0$. We can then already calculate the change in the terrain height at the particular point:

$$\Delta h = \begin{cases} -d_{bk} - \alpha * d_r & \text{if in riverbed,} \\ -\alpha * d_{bk} & \text{if on riverbank,} \end{cases} \quad (2.17)$$

where d_{bk} is the bank height and d_r is the river depth. However, subtracting this from the actual terrain height will lead to issues in locations where two rivers meet, as their effects will be added together. Additionally, as I mentioned in Section 2.2, when we generate a river spline against the terrain gradient, we force every point of the spline to not exceed in height any previous point so that during the actual terrain generation, we can force the river to create canyons instead of flowing uphill. For this reason, we calculate the final elevation as follows:

$$h = \min(p_{ws|y}, \min(h_o, p_{sp|y}) - \Delta h), \quad (2.18)$$

where $p_{ws|y}$ represents the elevation of the generated point including other, already generated rivers, h_o the elevation without any rivers, and $p_{sp|y}$ the elevation of the closest point on the spline. This height is then be passed back to the game engine code, which applies it to the terrain heightmap.

As for the braiding or meandering rivers, we can simulate both cases by simply offsetting the riverbed from the spline. Intuitively, to create a meandering river, we only need to adjust the point's distance from the spline y . To create a braiding river, we generate multiple smaller channels and pick the one closest to the generated point. We then set y to be the distance from the center of this channel instead of the spline. As the river width, we then use the width of the channel instead of the width of the whole river.

2.4.1 Meandering rivers

The tool can generate two types of meanders: sinoid meanders and tightly-looped meanders. Since a riverbed can be offset left or right by adding or subtracting from y , generating sinoid meanders can be done by adjusting it to $y = y + \sin(x)$, where x is the distance from the source along the river. This will create a river that will follow a sinoid wave oscillating around the rivercourse spline.

This approach, however, leads to a problem - different control points can be assigned different meander wavelengths. An intuitive solution would be to use $\frac{x}{l_m}$ instead of x as the sine input, where l_m represents the meander wavelength. However, this could cause issues in segments at which the wavelength decreases between two control points since despite x increasing slightly, $\frac{x}{l_m}$ would decrease massively.

For this reason, we substitute x with a different value. Before starting terrain generation, we prepare an alternative array of distances dm , where for every tessellated point tp_i :

$$dm_i = \begin{cases} dm_{i-1} + \frac{x_i - x_{i-1}}{l_m} & \text{if } l_m > 0, \\ 0 & \text{otherwise.} \end{cases} \quad (2.19)$$

We can then use $y = y + \sin(\text{lerp}(dm_{i-1}, dm_i, x_{tab}))$ to compute the offset at any point of the river. If we are transitioning from a river with meanders to a river without meanders (i.e., when one control point has the wavelength 0 and its neighbor does not), we only interpolate the meander width parameter while keeping the wavelength constant at the non-zero value. However, one detriment of this approach is the dependency of dm_i on the value x and all values l_m in previous control points after the last previous point with $l_m = 0$. As such, changing a single point at which the river meanders requires us to regenerate the rest of the meandering section of the river following the changed point.

The other type of meanders that can be generated are the tightly-looped ones. Compared to sinoid meanders, their main distinction is that these can cause the river to shortly flow against the direction of the valley it occupies. Also, while we expect that for any distance along the spline x , we can determine a single offset y through which the river flows, this is not the case for meandering rivers, where the river can pass through three offsets (see Figure 2.4).

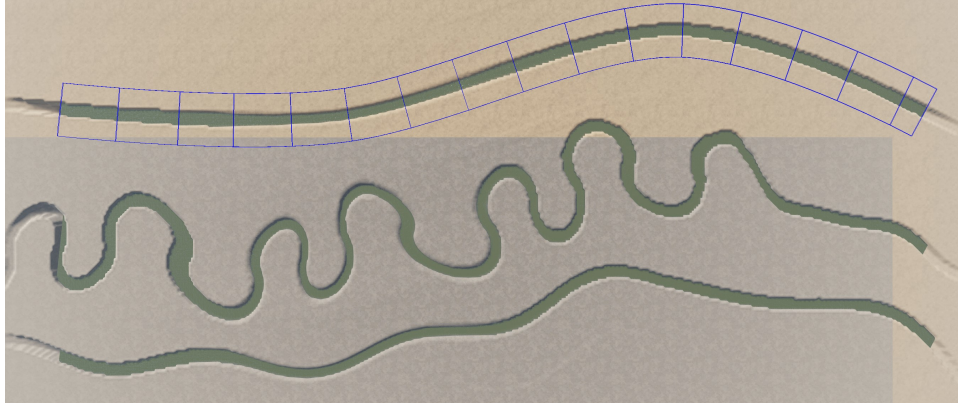


Figure 2.4: Three rivers created by identical splines: no meanders on top, tightly-looped meanders with randomness in middle (with the right end of the river transitioning to no meanders), sinoid meanders on bottom.

As such, the following process is used to create tightly-looped meanders instead:

1. Along each side of the spline, create an infinite sequence of circles

$$O_1 : c = [t * p, w_m], r = 0.5 * l_m, \quad (2.20)$$

$$O_2 : c = [(t + \frac{1}{2}) * p, -w_m], r = 0.5 * l_m, \quad (2.21)$$

where w_m is the meander width calculated as the product of meander wavelength and meander width ratio, and p is the meander period calculated as $2 * l_m - 2 * \sqrt{l_m^2 - w_m^2}$. This creates a sequence of circles, where each circle touches its two neighbor circles from the other side of the spline, the point of contact is always on the spline, and circles do not intersect as long as two can fit into the period ($p > 2 * l_m$). See Figure 2.5.

2. We use the values $[mod(x, p), y]$ as coordinates to find the distance to the nearest circle circumference (which can then be used as a substitute for y). However, this would carve the whole circles into the terrain, instead of just their outer curves - for

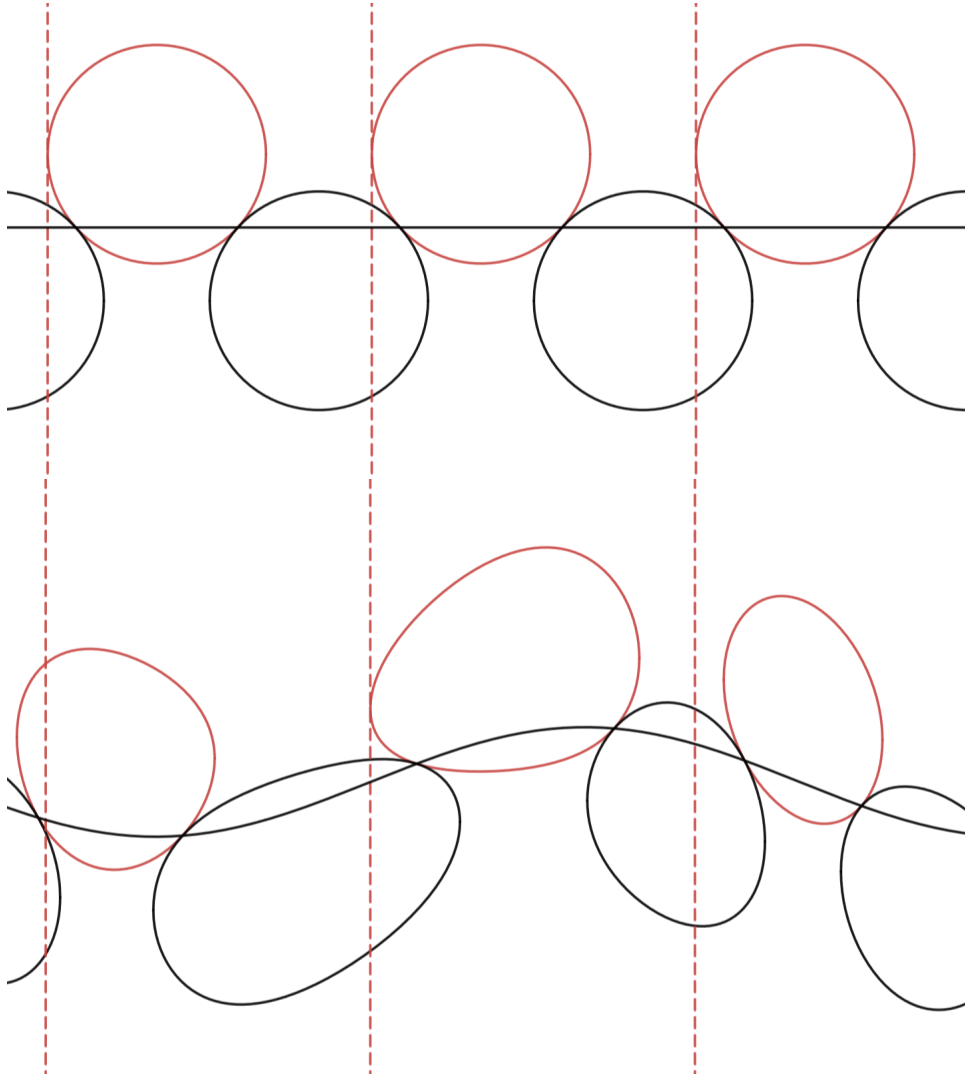


Figure 2.5: Circles created by the tightly-looped meanders algorithm. Red vertical lines denote the period boundaries, black horizontal line represents the X-axis. When evaluating a point above the X-axis, we ignore the circles with center below it, and vice-versa, to create a single tightly-looping line. In the image below, the effects of warping the input coordinates can be seen - by warping the X-axis together with the circles, we keep the constraint that the circles can only touch at the X-axis.

this reason, we will always only consider circles whose center is on the same side of the spline as the generated point (O_1 if $y \geq 0$, O_2 otherwise).

3. To add variety, we set

$$rd = l_m * K_r * \sin\left(\frac{x}{K_r}\right), \quad (2.22)$$

where K_r denotes the **River meander randomness** parameter set by the user with a value within $[0, 1]$. We then warp the input coordinates by changing them to $[mod(x + rd, p), y + rd]$. By only warping the input, the circles and the spline do not change in relation to each other, and the constraint that circles can only touch each other at the spline is kept, as well as the constraint that the circles cannot intersect as long as $p > 2 * l_m$.

As can be seen in Figure 2.4, this method can, despite its simplicity, create visually pleasing tightly-looped meanders.

At the same time, however, our approach of creating meanders by significantly offsetting the river from the spline does cause issues that, in the end, make it unsuitable for our use case. I further discuss these issues and a related solution in Subsection 4.1.3.

2.4.2 Braiding and anastomosing rivers

For braiding rivers, I use a similar approach to creating sinoid meanders. However, even with different parameters, using a sinoid function for every channel would create a very artificially looking river. For this reason, I have decided to use the Simplex noise function, a variation of Perlin noise, instead. The main benefit of the Simplex noise over Perlin noise is its probability distribution function. Perlin noise's curve follows a bell curve with very narrow tails - as such, channels would mostly oscillate around the spline and only rarely wander to the edges of the river bank. On the other side, Simplex noise distribution has a broad head with fatter tails, leading to individual channels being dispersed along most of the width of the riverbed [36, 37]. The presented tool uses a slightly modified version of a 1D Simplex noise implementation by Gustavson [38].

As braiding rivers consist of channels of various sizes, I divide the full width of the river between the channels as:

$$w_i = w_r * (\log_2(1 + l_1) - \log_2(1 + l_0)), \text{ where} \quad (2.23)$$

$$l_0 = \frac{i}{n_c}, \quad (2.24)$$

$$l_1 = \frac{i+1}{n_c}, \quad (2.25)$$

w_i is the width of channel i , and n_c is the channel count. This creates an ordered set of channels, where a channel with a lower index is always wider than channels with higher indices, and the sum of widths of all channels is equal to the river width w_r . This does not completely match the behavior of braided rivers in reality, where braiding often results in wider but shallower channels and where branching of a single channel does not affect the width of channels on the other side of the riverbed. However, the braiding pattern looks realistic enough for the tool's purpose. Then, for each channel, we sample the Simplex noise function, adding two layers of noise - one with larger influence and larger wavelength, which determines the overall position of the channel in the riverbed, and one with smaller influence and lower wavelength, which adds further detail.

An important thing to notice is that if we used the distance along the spline as the parameter for the Simplex noise, we would again run into the same issue as with sinoid meanders. However, we can instead use the same approach as we do with sinoid meanders, where, for input, we use the distance from start adjusted by the combined width of the banks $w_{all} = w_{bk|right} + w_{bk|left}$, computed by reusing the Equation 2.19, with l_m substituted with w_{all} .

However, this process alone only creates a set of channels that, under closer inspection, look detached from each other and do not seem to be a part of a single riverbed. One issue is the blending of appearing and disappearing channels. As I mentioned in Section 2.3, I have decided to represent the channel count parameter by real numbers instead of integers, using the fractional part as the blending parameter. This blending parameter can be used to interpolate the last channel's position between the value of its Simplex noise and the position of the previous channel in the channels sequence.



Figure 2.6: An anastomosing river consisting of up to 6 channels on a plain.

The second issue is that in some parts of the river, all channels can collapse close to the center of the spline, limiting the final river width to the width of the largest channel. To solve this, we compare the positions and extents of all channels, and if we find that two channels overlap, we transfer some of the width of the smaller channel to the larger channel.

Finally, the last step is to choose the channel that will create the deepest riverbed at the generated point by minimizing $\frac{|y - y_{c_i}|}{w_i}$, where y_{c_i} represents the offset of channel i . We then replace the w_r with w_i and continue with the rest of the carving algorithm as if the channel itself was the single river channel.

3 Implementation in Enfusion Engine

The previous chapter described the generating process focusing on the rivers. While this should give the reader an idea of how the proposed method can be implemented in a game engine, it conceals the more technical aspects of such implementation. Therefore, in this chapter, I will focus on how the underlying game engine code enables us to create terrain generators in the script that react to world changes, as well as on maintaining reasonable time and memory requirements.

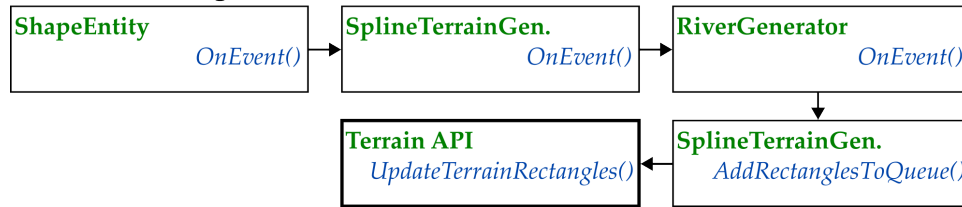
3.1 Terrain generation in Enfusion Engine

To keep the amount of work required to implement the river generator at a reasonable level, I made use of as much of the existing terrain API as possible. `Enfusion Engine` already contains a `RoadGeneratorEntity` class, whose purpose is to create road embankments along roads defined by splines. However, a few of its shortcomings make it nonviable for our use case, such as that it is not expandable by script, it often regenerates the entire length of the road even if only a single point has changed, and it often regenerates a single heightmap coordinate multiple times even after a single change. Overall, it focuses more on generating a single point without acknowledging the hierarchy in which the entire feature is divided into multiple patches consisting of a rectangle of heightmap points. Together, these issues lead to poor performance.

Despite these shortcomings, the similarities between the objectives of `RoadGeneratorEntity` and `RiverGeneratorEntity` made it a good source of inspiration for what could be improved by the implementation of `SplineTerrainGeneratorEntity`.

As I discussed in Section 1.1, `Enfusion Engine`, like most other engines, uses heightmaps to describe terrain elevation data. Modifying the terrain equates to swapping a specific part of the heightmap with modified pixels. When we adjust the terrain using sculpting tools, for instance, a rectangle of heightmap within the brush size is taken out, each pixel is modified based on the brush strength and mask, and the result is inserted back into the entire heightmap. However, when generating the terrain along a spline entity, this approach would be

On river change:



On next frame:

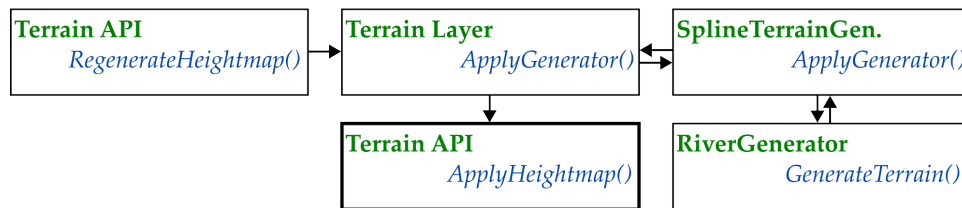


Figure 3.1: The steps of terrain generation after a change in the spline represented by an event on the ShapeEntity class. RiverGenerator prepares a set of terrain patches to update that is then passed to the terrain API. On the next frame, terrain API lets the terrain layers, and in turn, generators, regenerate the affected heightmap.

inefficient, as the bounding box of the entire spline will often contain many points, which the terrain feature will not influence. For this reason, the tool processes the spline segments, as divided by the control points, independently. This also allows us to filter which segments should be regenerated - for instance, for a river defined by a Catmull-Rom spline, moving a control point can not affect more than the next two spline segments on each side of it. After deciding on which terrain patches are to be regenerated, we simply use the terrain API to request the heightmap update.

The heightmap can also consist of multiple layers. The base layer contains the imported heightmap and any changes done by the user with sculpting tools. Further layers can then be created for different features, such as roads or rivers. Most importantly, this allows us to revert any changes done by the river generator, as well as blend merging rivers correctly (as described in Section 2.4 by using the elevation value h_o , which disregards the rivers layer of the heightmap). The terrain representation seen in the World Editor and the game is then a combination of all those layers.

The engine terrain implementation then controls the regeneration itself. First, the terrain patch is extracted from the base layer of the original heightmap. Second, if the regeneration was triggered by a sculpting tool, its effect is added to the base layer. Third, all additional layers are updated - in the case of the generated layers, like the `RoadLayer` and the `SplineTerrainGeneratorLayer`, all active generators, whose bounding box of influence overlap with the modified patch, are evaluated, and changes added by those generators are saved into the heightmap layer. Finally, the layers are combined, and the new terrain patch is inserted into the heightmap representation in memory and uploaded to the GPU.

While this approach brings the advantages mentioned above, it makes the process rather demanding on the optimization of the generators. If a terrain patch is to be regenerated, every river that could theoretically flow through that patch needs to be reevaluated within the patch's borders. At the same time, it is often just the spline entity's bounding box that overlaps with the terrain patch - the river itself does not flow through it. As such, it is important that at every step of the terrain generation process, we evaluate if applying a particular river generator on a particular heightmap patch or heightmap pixel is needed, as early exits can shave off a large part of the time needed to run the tool.

3.2 Applying the terrain changes

An important information that helps us optimize the process of generating terrain is the extent of influence of the generator surrounding the spline. For a river, this is usually the sum of half the river's width and the width of the larger of the river's banks. If the river has meanders, this value increases by the maximum possible width of the meanders. As such, this value can be easily derived from the cross-section parameters for every spline segment. We can also create an axis-aligned bounding box of a spline segment as an AABB of all the points gathered by tessellating the segment itself, expanded in all directions by the segment's possible extent. Thus, we arrive at the final bounding box containing all the heightmap pixels that the river segment can influence. By combining the bounding boxes of all segments, we get

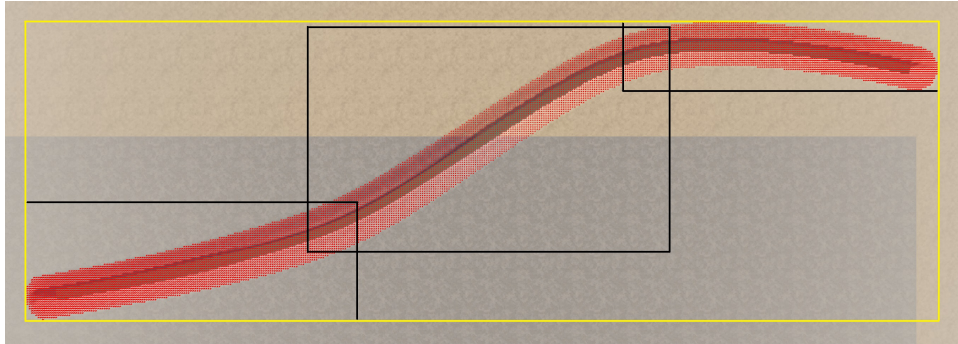


Figure 3.2: The extent of terrain changes performed by a single river consisting of three segments (four control points). The yellow rectangle represents the bounding box of the entire river, the black rectangles represent the bounding boxes of spline segments. Finally, the red points are points for which the `GenerateTerrain()` function was called.

the bounding box of the entire river. For memory conservation reasons, this is also the only value we keep even if the terrain is currently not being generated.

Keeping the bounding box of the entire river is also important for another reason - it lets the terrain implementation know when to execute our generator after any terrain changes done by the user or other generators.

When a new terrain patch is added to the queue for regeneration, the game engine can start by comparing the terrain patch rectangle and the river bounding box (yellow rectangle on Figure 3.2). If there is an overlap, the river generator can prepare additional data. We compute the tessellated points of the spline, the function $tcp(tp)$ transforming a tessellated point index to control point index, spline segment bounding boxes, distances from the beginning of the river, and create an empty 2D bitfield with the dimensions of the river's bounding box. Since terrain patches requested for regeneration are defined by rectangles, they often overlap, which leads to a heightmap pixel being evaluated by the same generator multiple times with the same result. To prevent this, we will use the bitfield to track which points were already generated and thus can be skipped. We keep this structure until the terrain

patches queue has been emptied, i.e., until all terrain generating is done.

After that, the terrain implementation generates each terrain layer, applying the spline terrain generators. At this step, we can finally evaluate which heightmap points should be influenced by which spline segments and call the scripted generator implementations to calculate the new elevations. For every point, we start by checking the associated bit in the bitfield - if it is true, the point has already been computed, and we can continue to the next point. Otherwise, we set the bit to true now and continue to the next step. We now need to find the spline segment closest to the generated point. Since we have created an array of bounding boxes for each segment (black rectangles on Figure 3.2), we can use it to disregard any segments which certainly have no influence on the heightmap point. For the remaining segments, we take their tessellated segments (prepared in the structure at the beginning of terrain generation) and compute their distance from the point. We select the tessellated segment closest to the point and check if the distance is less than the extent of the original spline segment (which we can get using the $tcp(tp)$ function). Finally, if this is true, we determine whether the point is on the left or right side of the spline (to compute the signed distance from the spline, which we pass to the scripted generator implementation and use to create non-symmetrical riverbeds) and compute the position on the spline closest to the generated point. We can then pass these information to the `GenerateTerrain()` script function, which calculates the final terrain evaluation as described in Section 2.4. Points for which the function was called are visualized in Figure 3.2 as red dots. This process is repeated for every point in the terrain patch, for every river, and finally, for every terrain patch in the queue.

3.3 Performance analysis and optimization

To measure the performance of the tool, I have used a 32x32 km map with a 2m planar heightmap resolution, on which I created 10 rivers of various lengths.

The spline shapes of the rivers and their parameters were all created by the tool automatically - I only interfered with the shape gen-

Table 3.1: Lengths, segment counts, and bounding box areas of rivers tested during profiling.

River	Length [m]	Spline segments	AABB area [km ²]
R_{01}	1772	37	1.05
R_{02}	2192	40	1.12
R_{03}	4770	78	6.92
R_{04}	6186	169	6.36
R_{05}	7221	125	12.57
R_{06}	8444	237	15.68
R_{07}	8634	234	19.89
R_{08}	10 554	336	29.56
R_{09}	24 585	555	120.79
R_{10}	27 611	864	142.81

eration when necessary due to the shape generation being stuck in a local terrain minimum.

As the objective of this thesis is to create a tool that can be used by environment designers in the process of creating a terrain for a game, and one of the essential constraints is that they should be able to quickly adjust the results and iterate over them, it is necessary for the tool to generate the terrain fast enough. While real-time generation would be preferred (and is, in fact, attainable, as many related works discussed in sections 1.1 and 1.3 have proven), this would require us to perform the terrain generation on the graphics processing unit (GPU). However, this is not possible as long as the users are supposed to be able to create custom generators using the game engine's scripting language. As such, I have had to focus on optimizing the algorithm for it to run sufficiently fast on the CPU.

Using the profiling tool Tracy¹, which Enfusion Engine directly supports, to determine the most time-consuming tasks, it was clear that the two most demanding actions were the calls to the `GenerateTerrain()` script function, which generates an elevation for a particular heightmap point (comprising around 85% of the time spent by the river generator), and computing the distances between tessellated segments of

1. <https://github.com/wolfpld/tracy>

the spline and the heightmap points when determining which points should be influenced by which segment (around 14%).

Since the `GenerateTerrain()` script function can be programmed not only by game developers but also by modders and other players, it is not possible to depend on them to ensure its optimization. This led me to the implementation of the optimizations described in the previous section, where I try to limit the amount of `GenerateTerrain()` calls the tool does to a minimum required for its proper functioning.

After implementing these optimizations, I have also compared the amounts of saved `GenerateTerrain()` calls and computations of point-to-line segment distances. Results are in tables 3.2 and 3.3.

Table 3.2: Decrease in `GenerateTerrain` calls required for generating each river. Point regeneration requests, requests denied by bitfield comparison, requests denied by final segment extent check, and actually performed calls (all counts are in thousands). The last column contains the fraction of unnecessary requests skipped due to the optimizations.

River	Req.	Den. bf.	Den. ext.	Perf.	Reduction
R_{01}	80.0	26.3	41.2	12.4	84.5%
R_{02}	82.3	23.0	49.2	10.2	87.6%
R_{03}	292.3	100.0	148.7	43.4	85.1%
R_{04}	335.2	141.6	145.2	48.4	85.6%
R_{05}	346.1	105.1	188.4	52.6	84.8%
R_{06}	716.8	365.5	225.7	125.6	82.5%
R_{07}	670.4	348.8	196.4	125.3	81.3%
R_{08}	847.3	438.0	274.4	134.8	84.1%
R_{09}	2 022.6	996.9	644.5	381.3	81.1%
R_{10}	3 237.5	1 952.2	757.3	527.9	83.7%

Finally, I have also measured the difference in the time required by the tool to generate those rivers, including comparing the optimization's effects. To ensure that the rivercourse generation would generate a river as similar as possible to the rivers in Table 3.1, I had the tool generate the rivercourses with the rivers already carved into the terrain. Since this does not guarantee an accurate comparison, the times required for rivercourse generation should only be considered

Table 3.3: Decrease in line segment/point distance computations required for generating each river thanks to segment bounding box comparisons. All counts are in thousands. The last column contains the fraction of unnecessary requests skipped due to the optimizations.

River	Requested	Performed	Reduction
R_{01}	1 983	33	98.3%
R_{02}	2 376	33	98.6%
R_{03}	14 990	146	99.0%
R_{04}	32 723	135	99.6%
R_{05}	30 115	163	99.4%
R_{06}	83 254	378	99.5%
R_{07}	75 280	350	99.5%
R_{08}	137 505	426	99.7%
R_{09}	569 314	1 139	99.8%
R_{10}	1 110 503	2 030	99.8%

indicative. These results can be found in Table 3.4. The times were measured on a computer with an Intel Core i7-8700K processor, nVidia GeForce GTX 1070 graphics card with 8GB VRAM, and 16GB RAM clocked at 2.4GHz. The terrain data were saved on an SSD storage device. However, as the engine loads the entire heightmap into memory before processing it, this should not significantly influence the speed of the terrain generation itself.

3.3.1 Regenerating rivers after changes

While the times in Table 3.4 are acceptable for the initial generation of the whole river, it is essential that the user is not forced to wait through the entire regeneration every time they make a slight adjustment to the river. As updating the terrain is done in patches, we can easily reduce the time needed to update the terrain by reducing the number of patches we request for updates with each change. For this reason, I have allowed the `SplineTerrainGeneratorEntity` to receive events from spline tools, which allow it to specifically react to any changes in the spline or the river parameters. After each event, we can react accordingly - for instance, if a point is added, moved, or deleted, we

Table 3.4: Times (in seconds) required to process each of the terrain generation steps. The times in the first three columns are averages of three attempts of rivercourse shape generation, parameters generation, and adjusting the heightmap. The times in the last column, which represent the last step without discussed optimizations, are a result of a single attempt, as they are not meant for comparison with other similar tools. These times measure only the time spent by the river generator - actual times, as perceived by the user, can be 0-2 seconds longer due to the processing of the modified heightmap by the terrain implementation.

River	Shape	Params	Carving (opt.)	Carving (not opt.)
R_{01}	0.36	0.16	0.6	2.9
R_{02}	0.33	0.17	0.4	2.6
R_{03}	0.54	0.19	2.4	11.9
R_{04}	0.84	0.24	4.8	26.0
R_{05}	0.92	0.21	5.3	25.8
R_{06}	1.41	0.21	6.6	28.4
R_{07}	1.98	0.27	6.4	25.9
R_{08}	3.31	0.33	7.2	35.6
R_{09}	6.27	0.46	18.9	87.0
R_{10}	6.05	0.62	30.3	162.2

only need to regenerate the area surrounding the two segments above and below it. In this case, it is also important to regenerate not only the new bounding boxes of those segments but also the old ones so as not to keep old sections of rivers in the heightmap. If the river parameters of a control point are changed, we need to regenerate the river between the previous and next control points, that were assigned their own set of parameters. If any of this is done to a point that has meanders or multiple channels, we also need to regenerate all following points until reaching one that does not contain the particular feature (this is due to us using specialized inputs for the sine and Simplex noise functions used by those features).

As such, the entire river needs to be regenerated only if it was previously disabled or if the entire spline was moved at once.

To further decrease the time needed for regenerating terrain, we also allow the user to temporarily pause terrain regeneration for a particular river. In such a case, we add the patches that need regeneration to a separate array and only submit them to the terrain implementation once the user unpauses the terrain regeneration. This is useful if the user wants to make changes to multiple points at once and does not require immediate visual feedback.

4 Discussion

4.1 Shortcomings and possible improvements

As this thesis proposes a novel method for the procedural generation of rivers for video game environments in video game engines, this section will focus on evaluating the suitability of developed features for fulfilling the tool's objectives and propose possible options for improving them. As the tool was also evaluated by members of Bohemia Interactive's environment design department, this list includes both the author's and their propositions.

4.1.1 User experience

Using a procedural generator instead of carving the rivers manually should have two main benefits: decreasing the person-hours required to create the rivers and mitigating mistakes that a person could make in the process. As such, the tool should generate the initial river as fast as possible and allow the user to iterate changes to the river shape and parameters quickly.

In this sense, one of the main shortcomings of the tool is that if river parameters are generated automatically, they are created for every control point. If the user wishes to modify a whole section of a river, they need to manually edit the value of the parameter at every control point, which is both time-consuming and can lead to mistakes.

Two solutions are available for this issue. Firstly, in its current state, the tool generates and saves the parameters for each control point, which is inefficient. In many cases, parameters of control points could be easily interpolated from their neighbors, especially in straight segments where no meanders, braiding, and sharp curves are present. This is already supported in the third step of the generation, as the tool does interpolate parameters for points that do not have them set, so it would only be necessary to implement an algorithm that would evaluate which points' data can be interpolated and thus can be deleted. Also, this change could be fully implemented in the scripted `RiverGeneratorEntity`. For now, the only way the tool tries to opti-

mize the whole structure's size is by removing unnecessary control points.

A second, more robust solution would be to implement a brush-like tool, where the user can set the parameter they want to change, the strength of the tool, and then brush over the segment of the river which should be modified. Since the speed of the river generation currently does not allow for real-time terrain adjustments, simple geometric shapes could be used to visualize changes in the river's parameters, with the changes being only applied after the user finishes using the brush. A rough implementation of this method could also be written entirely in the script (script code can read the mouse cursor's position inside the World Editor viewpoint). However, a proper implementation would require a generic brush tool included in the in-game code, allowing the scripted generators to supply their own sets of parameters.

Another user experience related issue is that as the spline entity and the `RiverGeneratorEntity` are two separate objects, changing cross-section parameters and generator parameters is done in two separate windows. A UI element that would unify all the available options in a single place, such as a specific river tool, which would combine all the possible options in a single window, could help the user navigate them more efficiently. After the evaluation, environment design team members mentioned they often had problems finding out what options the tool provides. This included issues such as not understanding the purpose of the *Update Heightmaps* option (which is disabled by default, leading to an impression that completely disabling and enabling the generator is needed after every change in the river) and not knowing how to manually adjust the width of the river.

Implementing a simpler and more unified UI for the tool could be therefore expected to both increase the productivity of working with the tool as well as help the user find all the options that the tool offers.

4.1.2 Generating optimal rivercourse

One of the three main functions provided by the tool is the procedural generation of the rivercourse by creating a spline, which the river should follow. This is implemented by traversing the terrain down the gradient in steps of user-defined length. While I found the feature

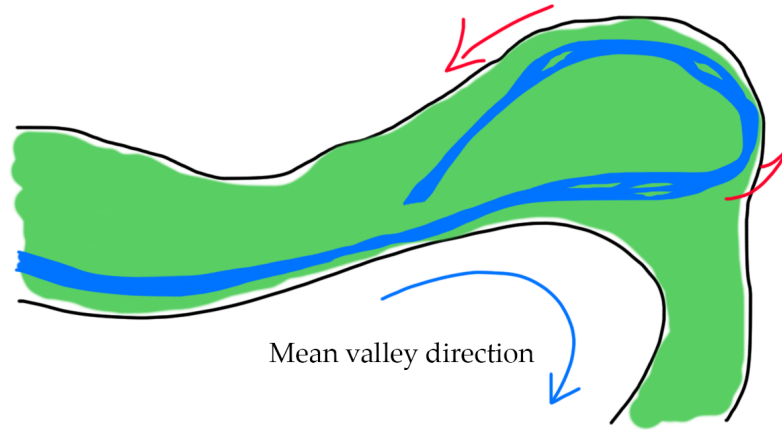


Figure 4.1: A formation of a loop in a valley with low mean slope caused by the lack of definition of the general river direction. The valley sides push the river against the mean valley direction, eventually forcing it to cross itself. Approaches based on shifting an existing spline are not prone to this issue.

to work quite well on terrains with varied elevations, lowlands often provide a challenge. If the gradient is not sufficient enough for the tool to determine a clear path downhill, the tool makes the river continue directly forwards on the reasonable presumption that the river should not change directions if there is no reason for it. However, in our implementation, even minor differences in elevation can have a significant influence on the further flow of the river - the generator can easily get stuck in local minima (which are often caused by the inaccuracies in the imported heightmap), and after hitting the edge of a valley, the river often completely changes its heading, sometimes even leading it to create a loop. While smaller loops sometimes occur in hilly terrains, too (and the tool automatically removes them), the lack of gradient in lowlands can cause the river to start flowing in a seemingly random direction.

The root of the problem lies in the inflexibility of the current iteration of the tool. When evaluating the score for candidates for the next

point of the river, the weight of other components of the score - the current heading and the penalty for points too close to the previous point - stays the same no matter the differences in the elevations of the candidate points. This can cause the generator to start adding the following points seemingly randomly. To fix this issue, it would be beneficial to adjust the influence of these components on the final score (see Equation 2.9) based on the differences in the elevations to ensure the gradient's main role in the score evaluation.

Alternatively, in Section 1.3.2, I have mentioned an implementation of a similar river generator tool, which also deals with adjusting splines to create rivercourses [34]. Instead of generating the spline from scratch, this tool allows the user to draw a rough outline of the river shape, which is then adjusted to make for a more realistic shape. Adding a similar functionality to the presented generator could prove helpful, as the user, if willing, could gain more control over the generation process, which is currently lacking.

4.1.3 Meanders

The procedural generation of meanders is essential to making realistically looking rivers. While some previous works offer basic river meandering, where a slight sinoid offset is added to the rivercourse before carving it into the terrain, I have decided to improve upon this technique by adding the possibility of creating larger meanders of two types - sinoid and tightly-looped meanders. In the proposed approach, the meanders are generated only in the third step, the carving of the river, and are not defined in the spline itself.

This, however, led to several issues. Affecting such a wide belt around the spline has a noticeable negative impact on the performance, as the area that is processed by the tool needs to be significantly larger than the area that is modified in the end. In addition, meanders change the rivercourse significantly, but the user only has minimal control over their exact position. Finally, if the user wants to add a water surface to the river, they must manually create a new spline that follows the meanders and assign the water surface to it.

For these reasons, I believe that while it is possible to allow the generator to add a small offset to the spline while carving the terrain as a way to increase detail, the generation of any larger meanders

should be instead handled in the first step, which is the rivercourse generation. After generating the initial shape, specific points of the river could be offset at the wavelength intervals by the meander width to create sinoid meanders, and calculating the period and radius of the circles forming them could be used to adjust the spline to create tightly-looped meanders. On the other hand, warping the initial coordinates to add variety to the meanders has proved a viable option that adds much-needed detail to them and could still be implemented in the last step of the generation process.

4.2 Further work

In this section, I will propose new features that could be added to improve the tool's usability without requiring significant changes in its design principles.

4.2.1 Cascading

In the description of the river types according to the Rosgen classification in Section 1.2, I have described the types Aa+ and A rivers as often consisting of various vertical features, such as waterfalls, cascades, and rapids. In its current form, the river generator is not able to create these features, which leads to mountainous streams flowing downhill at a constant gradient. Since this is not usual in natural rivers, it can often break the player's immersion and make the environment look bland. As such, adding this functionality to the presented method should be considered, especially as other works ([33, 34, 35]) have already presented viable methods for generating these features. Positions of cascades along the river could be generated as a part of the second step of the river generation process, the parameters generation. Then, during the heightmap generation, the generator could compare the elevations of the neighboring control points, and instead of interpolating between them linearly, larger cascades could be created. For rapids, the user could define elevation steps, and the tool would then use the smoothstep function to create a less linear gradient.

Despite this, it is technically possible to create cascades even in the current iteration of the tool by adding control points with tangents

whose slopes do not align with the mean slope of the river (see Figure 4.2).

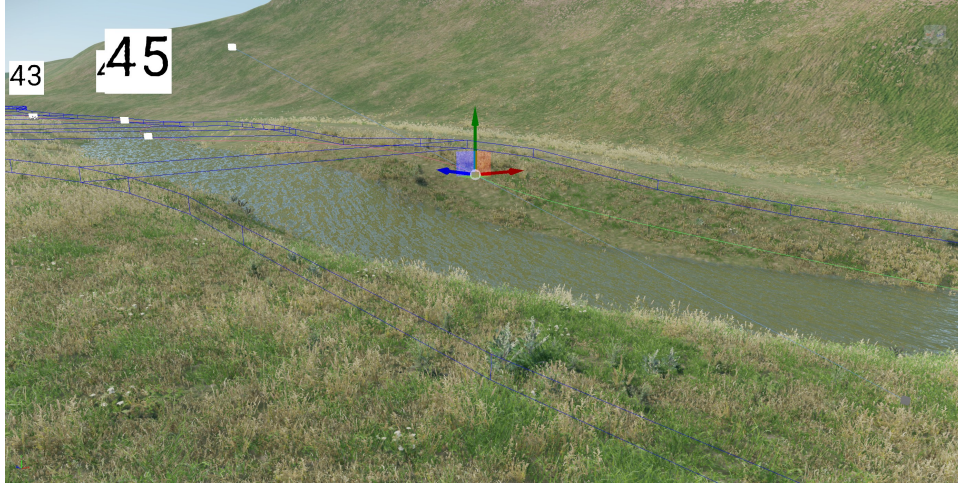


Figure 4.2: While the tool does not automatically create cascades, it is possible to create them by adding control points with a sloped tangent.

4.2.2 Confluences

While the third step of the generator, adjusting the heightmap, handles overlapping rivers (the river that is the deepest at a point overwrites the others), the first two steps do not. As a result, the spline generation will not end, resulting in two rivers sharing the same space instead of merging. Not only does the user then have to remove the excess points manually, but this also creates issues during the parameters generation. As most of our parameters are somewhat dependent on the flow volume of the river, the user then has to manually adjust the flow of the river below the confluence and generate the parameters again. Instead, the tool could detect the confluence of two rivers, merge the smaller river into the bigger one, and automatically raise the flow volume at the control points following the confluence. Additionally, while not an issue with the generator itself, it was noted by the environment designers that the current system often results in clipping of the river surface masks, making it challenging to create well-looking conflu-

ences. Therefore, improving the method for rendering river surfaces to allow the rivers to interact with each other would also be beneficial.

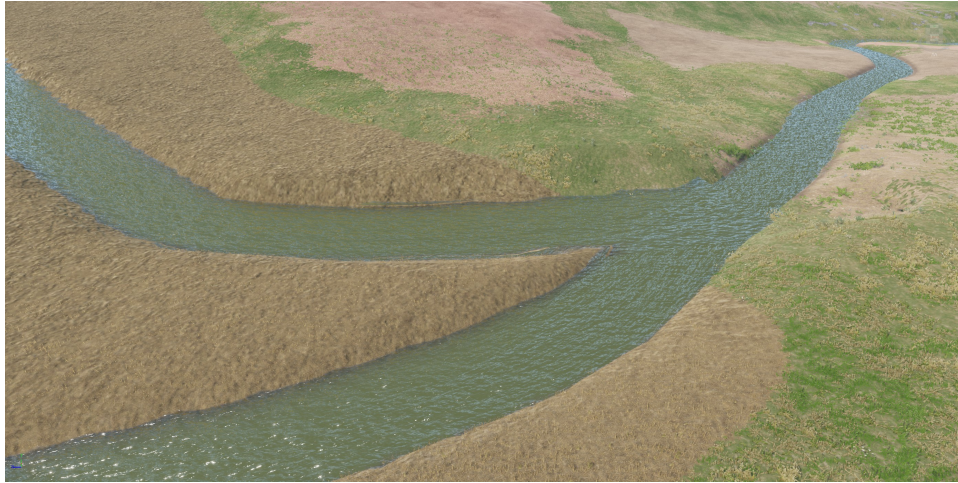


Figure 4.3: The tool currently does not create confluences automatically. A manual approach, where the user cuts off the overlapping parts of the river and aligns the rivers, often leads to subpar results, visible Z-fighting between the river surfaces, and the larger river not changing dimensions following the confluence despite the increase in water flow.

4.2.3 River surface and flow maps

In its current iteration, the river generator only performs changes to the terrain heightmap. However, this requires the user to manually add the river surface generator to the river to create the water mesh along the spline. Since the elevations provided by the spline generated by the tool are at the elevation of the original terrain, which is always higher than the desired river surface, the user then has to manually set the width of the mesh and the elevations of its points.

While adjusting the vertical offset at each control point is possible and could be considered an easy solution, changes in elevation between control points are often not gradual. As such, the height difference between the spline and the underlying terrain can vary wildly throughout a single segment. For this reason, a separate spline

is needed for the river surface, where the height differences between the points are also considered during the spline optimization process.

As this is the only step in creating a river that is still necessarily done manually by the user, automatically generating this spline and setting the width of the segments on the river surface generator would make the tool feature-complete.

To improve the river surface generation, the tool could automatically export information on flow direction, velocity, and turbulences caused by the cascades and rapids and save them as a flow map. As this is already supported by the material used in the Enfusion engine for creating river surfaces, all that is required from the tool is to create the needed texture. Since the presented method already evaluates the heightmap texture on a point-after-point base, expanding it to simultaneously alter the flow texture should be possible without much effort.

4.2.4 Terrain textures and vegetation

When a river flows over soil, its composition often changes. This results in riverbeds consisting of gravel in the mountainous regions, mud and sand in the lowlands, as well as the formation of sediment bars made from these materials where sediment deposition occurs. Changing the texture of the terrain is currently something the designer has to do manually after adding the river. This process can take upwards to a few hours for even a medium-sized river (approximately 10 kilometers). Nevertheless, the terrain texture generation could be done automatically by the tool, as it only requires us to make trivial changes to a set of textures, similar to how flow maps could be handled.

Finally, while this thesis has mainly discussed rivers as a geographical feature, their other important role is the formation of unique ecosystems. Many species of plants and trees can only be found around rivers, with their distribution often influenced by the river's depth, water and soil composition, or flow speed. Based on this, the user could create rules that govern the spawning of vegetation alongside river channels, with the tool then generating the population automatically. Other objects, such as boulders or trunks of dead trees, could also be automatically generated in mountainous streams that have sufficient flow to carry them.

Conclusion

The objective of this thesis was to propose and implement a method for the automatization of carving riverbeds into existing terrains in the Enfusion Engine. Additional requirements the tool should meet were discussed with the environment design department of Bohemia Interactive and were considered during the design and development process.

The river generator tool was implemented and is currently a part of the Enfusion Workbench World Editor. At the time of publication, the tool is only available in the internal builds of the Workbench. However, it is expected to become part of the public versions of Workbench bundled with the Arma Reforger game during the first quarter of 2023.

The tool allows users to generate rivers in three steps. First, the user can define the shape of the river, either by manually creating a spline along the expected rivercourse or by letting the tool automatically create one based on the topography. Second, the user can either manually set the properties of the river or, again, let the tool generate them based solely on the expected water discharge of the river and the probability of the formation of multiple channels. In the last step, the tool automatically modifies the terrain heightmap to sculpt the river into the terrain. After the river is added to the terrain, removing it and easily reverting any changes is possible. Adjusting the parameters of an otherwise finished river is also possible and leads to regenerating only the necessary parts of the terrain.

The tool's implementation is divided into two parts - a generic `SplineTerrainGeneratorEntity` class, which handles interaction with the terrain API, and the `RiverGeneratorEntity` inheriting from it, which implements features specifically related to rivers. After discussing the possibilities of `SplineTerrainGeneratorEntity` with both the environment design as well as the Enfusion programming teams, there are plans to add additional generators based on this entity, such as a new `RoadEmbankmentGeneratorEntity`, which will create road and rail embankments, as well as a `LakeGeneratorEntity`, which will create lakes. Improvements to the `RiverGeneratorEntity`, many of which were discussed in Chapter 4, are also planned and will be im-

plemented by the author of the thesis during his further involvement with the company. Since `SplineTerrainGeneratorEntity` allows for generators to be implemented in the engine's script language, members of the Arma Reforger modding community will also be able to create custom generators of any other geographical features.

The current version of the tool can be tested using the Workbench build available in the electronic attachments and is also presented in the attached showcase video. Additionally, a set of screenshots showing different sections of three rivers created with the tool are also available.

Bibliography

1. SOCHOR, Jiří. *Terrain Rendering*. Masaryk University, 2020. Available also from: https://is.muni.cz/auth/el/fi/jaro2020/PA213/um/PA213_2020_Terrain_Rendering_4.pdf.
2. KAMAL, K. Raiyan; UDDIN, Yusuf Sarwar. Parametrically Controlled Terrain Generation. In: Perth, Australia: Association for Computing Machinery, 2007, pp. 17–23. GRAPHITE '07. ISBN 9781595939128. Available from doi: 10.1145/1321261.1321264.
3. FOURNIER, Alain; FUSSELL, Don; CARPENTER, Loren. Computer Rendering of Stochastic Models. *Commun. ACM*. 1982, vol. 25, no. 6, pp. 371–384. ISSN 0001-0782. Available from doi: 10.1145/358523.358553.
4. MANDELBROT, Benoit B.; VAN NESS, John W. Fractional Brownian Motions, Fractional Noises and Applications. *SIAM Review*. 1968, vol. 10, no. 4, pp. 422–437. Available from doi: 10.1137/1010093.
5. PERLIN, Ken. An Image Synthesizer. 1985, vol. 19, no. 3, pp. 287–296. ISSN 0097-8930. Available from doi: 10.1145/325165.325247.
6. KAMAL, K. Raiyan; UDDIN, Yusuf Sarwar. Parametrically Controlled Terrain Generation. In: Perth, Australia: Association for Computing Machinery, 2007, pp. 17–23. GRAPHITE '07. ISBN 9781595939128. Available from doi: 10.1145/1321261.1321264.
7. OLSEN, Jacob. Realtime Procedural Terrain Generation. 2004.
8. ONG, Teong Joo; SAUNDERS, Ryan; KEYSER, John; LEGGETT, John J. Terrain Generation Using Genetic Algorithms. In: *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. Washington DC, USA: Association for Computing Machinery, 2005, pp. 1463–1470. GECCO '05. ISBN 1595930108. Available from doi: 10.1145/1068009.1068241.
9. MUSGRAVE, F. K.; KOLB, C. E.; MACE, R. S. The Synthesis and Rendering of Eroded Fractal Terrains. In: *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1989,

- pp. 41–50. SIGGRAPH '89. ISBN 0897913124. Available from doi: 10.1145/74333.74337.
10. BENEŠ, Bedřich; FORSBACH, Rafael. Visual Simulation of Hydraulic Erosion. In: WSCG. 2002.
 11. ANH, Nguyen Hoang; SOURIN, Alexei; ASWANI, Parimal. Physically Based Hydraulic Erosion Simulation on Graphics Processing Unit. In: *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*. Perth, Australia: Association for Computing Machinery, 2007, pp. 257–264. GRAPHITE '07. ISBN 9781595939128. Available from doi: 10.1145/1321261.1321308.
 12. CHIBA, Norishige; MURAOKA, Kazunobu; FUJITA, Kunihiro. An erosion model based on velocity fields for the visual simulation of mountain scenery. *Comput. Animat. Virtual Worlds*. 1998, vol. 9, pp. 185–194.
 13. JÁKÓ, Balázs; TÓTH, Balázs. Fast Hydraulic and Thermal Erosion on GPU. In: *Eurographics*. 2011.
 14. BENEŠ, Bedřich; TĚŠÍNSKÝ, Václav; HORNYŠ, Jan; BHATIA, Sanjiv K. Hydraulic erosion. *Computer Animation and Virtual Worlds*. 2006, vol. 17, no. 2, pp. 99–108. Available from doi: <https://doi.org/10.1002/cav.77>.
 15. ŠTAVA, Ondřej; BENEŠ, Bedřich; BRISBIN, Matthew; KŘIVÁNEK, Jaroslav. Interactive Terrain Modeling Using Hydraulic Erosion. In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Dublin, Ireland: Eurographics Association, 2008, pp. 201–210. SCA '08. ISBN 9783905674101.
 16. MUSGRAVE, Forest Kenton. *Methods for Realistic Landscape Imaging*. USA: Yale University, 1993. Available also from: <https://www.kenmusgrave.com/dissertation.pdf>. PhD thesis. UMI Order No. GAX94-15872.
 17. BROWN, Rocko; PASTERNAK, Gregory; WALLANDER, Wesley. Synthetic river valleys: Creating prescribed topography for form-process inquiry and river rehabilitation design. *Geomorphol-*

- ogy. 2014, vol. 214, pp. 40–55. Available from doi: 10.1016/j.geomorph.2014.02.025.
18. MUSGRAVE, Forest Kenton. *Methods for Realistic Landscape Imaging*. USA: Yale University, 1993. PhD thesis. UMI Order No. GAX94-15872.
 19. ROSGEN, David L. A classification of natural rivers. *CATENA*. 1994, vol. 22, no. 3, pp. 169–199. ISSN 0341-8162. Available from doi: [https://doi.org/10.1016/0341-8162\(94\)90001-9](https://doi.org/10.1016/0341-8162(94)90001-9).
 20. BORGMAN, Christine L.; CHARLTON, Ro. *Fundamentals of Fluvial Geomorphology*. London: Routledge, 2007. ISBN 9780203371084. Available from doi: <https://doi.org/10.4324/9780203371084>.
 21. REICHWALDER, P.; JABLONSKÝ, J. *Všeobecná geológia*. Univerzita Komenského, 2003. ISBN 9788022316644.
 22. LEOPOLD, Luna B.; LANGBEIN, W. B. RIVER MEANDERS. *Scientific American* [online]. 1966, vol. 214, no. 6, pp. 60–73 [visited on 2022-11-11]. ISSN 00368733, ISSN 19467087. Available from: <http://www.jstor.org/stable/24930965>.
 23. LEOPOLD, Luna Bergere; MADDOCK, Thomas. The hydraulic geometry of stream channels and some physiographic implications. In: 1953.
 24. ALLEN, Peter M.; ARNOLD, Jeffrey C.; BYARS, Bruce W. DOWN-STREAM CHANNEL GEOMETRY FOR USE IN PLANNING-LEVEL MODELS1. *JAWRA Journal of the American Water Resources Association*. 1994, vol. 30, no. 4, pp. 663–671. Available from doi: <https://doi.org/10.1111/j.1752-1688.1994.tb03321.x>.
 25. MOODY, John A.; TROUTMAN, Brent M. Characterization of the spatial variability of channel morphology. *Earth Surface Processes and Landforms*. 2002, vol. 27, no. 12, pp. 1251–1266. Available from doi: <https://doi.org/10.1002/esp.403>.
 26. FRASSON, Renato; PAVELSKY, Tamlin; FONSTAD, Mark; DURAND, Michael; ALLEN, George; SCHUMANN, Guy; LION, Christine; BEIGHLEY, R.; YANG, Xiao. Global Relationships Between River Width, Slope, Catchment Area, Meander Wavelength, Sinuosity, and Discharge. *Geophysical Research Letters*. 2019, vol. 46. Available from doi: 10.1029/2019GL082027.

27. HOWETT, Julia. *Meander belt delineation: Developing a predictive model for meander belt width*. 2019. Available also from: <https://ir.lib.uwo.ca/etd/4915>. MA thesis. The university of Western Ontario.
28. CROSATO, Alessandra; MOSSELMAN, Erik. Simple physics-based predictor for the number of river bars and the transition between meandering and braiding. *Water Resources Research*. 2009, vol. 45, no. 3. Available from doi: <https://doi.org/10.1029/2008WR007242>.
29. BROWN, Rocko A.; PASTERNAK, Gregory B. How to build a digital river. *Earth-Science Reviews*. 2019, vol. 194, pp. 283–305. issn 0012-8252. Available from doi: <https://doi.org/10.1016/j.earscirev.2019.04.028>.
30. KELLEY, Alex; MALIN, Michael; NIELSON, Gregory. Terrain simulation using a model of stream erosion. In: 1988, vol. 22, pp. 263–268. Available from doi: 10.1145/54852.378519.
31. GÉNEVAUX, Jean-David; GALIN, Éric; GUÉRIN, Eric; PEYTAVIE, Adrien; BENEŠ, Bedrich. Terrain Generation Using Procedural Models Based on Hydrology. *ACM Trans. Graph.* 2013, vol. 32, no. 4. issn 0730-0301. Available from doi: 10.1145/2461912.2461996.
32. GÉNEVAUX, Jean-David; GALIN, Éric; GUÉRIN, Eric; PEYTAVIE, Adrien; BRIQUET, Cyril; GROSBELLET, François; BENES, Bedrich. Terrain Modelling from Feature Primitives. *Computer Graphics Forum*. 2015, vol. 34, no. 6, pp. 198–210. Available from doi: <https://doi.org/10.1111/cgf.12530>.
33. PEYTAVIE, A.; DUPONT, T.; GUÉRIN, E.; CORTIAL, Y.; BENES, B.; GAIN, J.; GALIN, E. Procedural Riverscapes. *Computer Graphics Forum*. 2019, vol. 38, no. 7, pp. 35–46. Available from doi: <https://doi.org/10.1111/cgf.13814>.
34. SIDEFX HOUDINI; SCHRIJVERS, Ben; LOPATION, Vladimir. *Guerilla Games | Horizon Zero Dawn* [online]. 2017-05 [visited on 2022-11-12]. Available from: <https://vimeo.com/216727778>.

BIBLIOGRAPHY

35. TOKAREV, Kirill; TRÜMPLER, Simon. *River Simulation in Houdini & UE4* [online]. Ed. by 80.LV. 2018-11 [visited on 2022-11-12]. Available from: <https://80.lv/articles/006sdf-river-simulation-with-simonschreibt/>.
36. CHEN, Rudi. *Restricted Perlin Noise for Better Rendering* [online]. Ed. by DIGITAL FREEPEN. 2017-06 [visited on 2022-11-21]. Available from: <https://digitalfreepen.com/2017/06/22/restricted-perlin-noise.html>.
37. NELSON, Minar. *Distribution of 2D noise functions* [online]. Ed. by NELSON'S LOG. 2021-10 [visited on 2022-11-21]. Available from: <https://nelsonslog.wordpress.com/2021/10/10/distribution-of-2d-noise-functions/>.
38. GUSTAVSON, Stefan. Simplex noise demystified. 2005. Available also from: https://www.researchgate.net/publication/216813608_Simplex_noise_demystified.
39. GLASSNER, Andrew S. (ed.). *Graphics Gems*. USA: Academic Press Professional, Inc., 1990. ISBN 0122861695.

A River generator manual

This appendix will describe how to test the tool using the Workbench build present in the thesis archive (available at <https://is.muni.cz/auth/th/dss2n/>). I will also provide an overview of the available parameters and how to access them.

Please note that the Workbench requires a DirectX 12 compatible graphics card to run and will fail to start when using an incompatible device. Furthermore, since Workbench is normally distributed as part of the Arma Reforger, some of its features require game data to work correctly. Since these data are not present in the thesis archive, these features might be limited. The river generator tool, however, is fully functional.

A.1 Starting the Workbench and loading a map

1. Using the `startWB.bat` batch file is recommended to start the Workbench. This file launches Workbench with the necessary data and the useful `-logLevel verbose` parameter. Otherwise, starting the `/bin/WorkbenchApp.exe` program with the working directory set to `/CoreData` is necessary.
2. To use the `RiverGeneratorEntity` and open the sample maps, the Workbench must load the correct project with all its dependencies. In the Enfusion Workbench Launcher window, click the *Add Existing* button in the bottom left corner and select the `/PackedData/GameTemplate.gproj` project file. Then, do the same with the `/RiverGenData/RiverGenerator/addon.gproj` file. Finally, select the `RiverGenerator` project and click *Open* (see Figure A.1).
3. Open the World Editor using the *Editors* → *World Editor* option in the menu bar or by clicking the large *World Editor* button in the center of the window.
4. Load one of the available maps using the *Load World* button in the toolbar (or by pressing `Ctrl+0`, see Figure A.2). Available maps are:

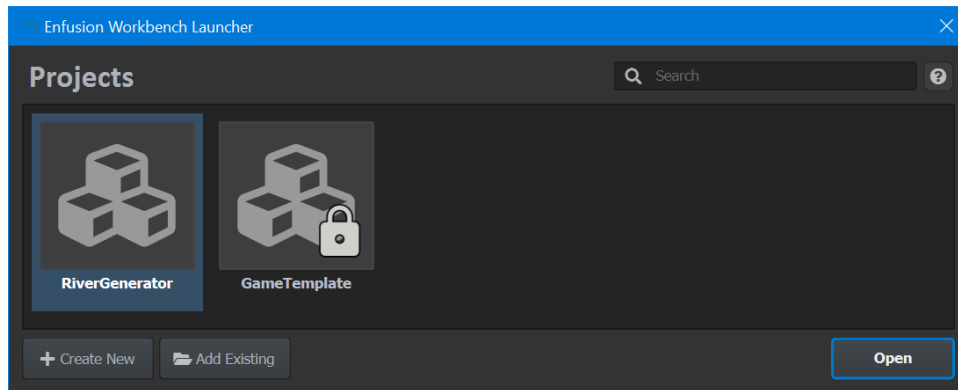


Figure A.1: The Workbench Launcher after adding the two necessary projects. Adding both projects is essential, as GameTemplate is a dependency of RiverGenerator.

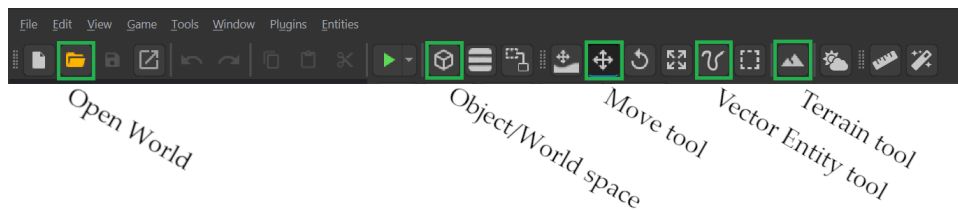


Figure A.2: Options available in the World Editor toolbar.

- *EmptyEden*, a slightly edited version of the Everon map present in the game Arma Reforger with purely generated rivers - the lack of human influence, however, usually makes for somewhat bland rivers,
- *EmptyEdenAdj*, a heavily modified version of the EmptyEden map with rivers with manually set parameters - it is specifically intended to showcase the larger branching river in the "delta" near the center of the island,
- *Hill*, a larger map intended to allow the user to test the tool on larger maps,
- *Mountain*, a larger map that showcases the tool on a terrain with extreme elevation variety, including the shortcomings of simply adding the river surface to the generated spline (discussed in Subsection 4.2.3),

- *CustomMap*, a semi-large map that allows the user to create a custom terrain (as the other maps cannot be manually edited, as they are compressed and read-only).
5. After loading the world, the user will be presented with the window depicted in Figure A.3. It is recommended to open the *Hierarchy* and *Log console* tabs (highlighted in green at the bottom of the window). The two tools most helpful in manipulating rivers are the *Transform* (hotkey W) and the *Vector Entity* (hotkey V) tools (highlighted in yellow in the toolbar on top of the window). Switching between the selected object properties and the selected tool options is possible using the tabs in the bottom right of the window (highlighted in orange). The error messages printed in the log console can be ignored, as they only

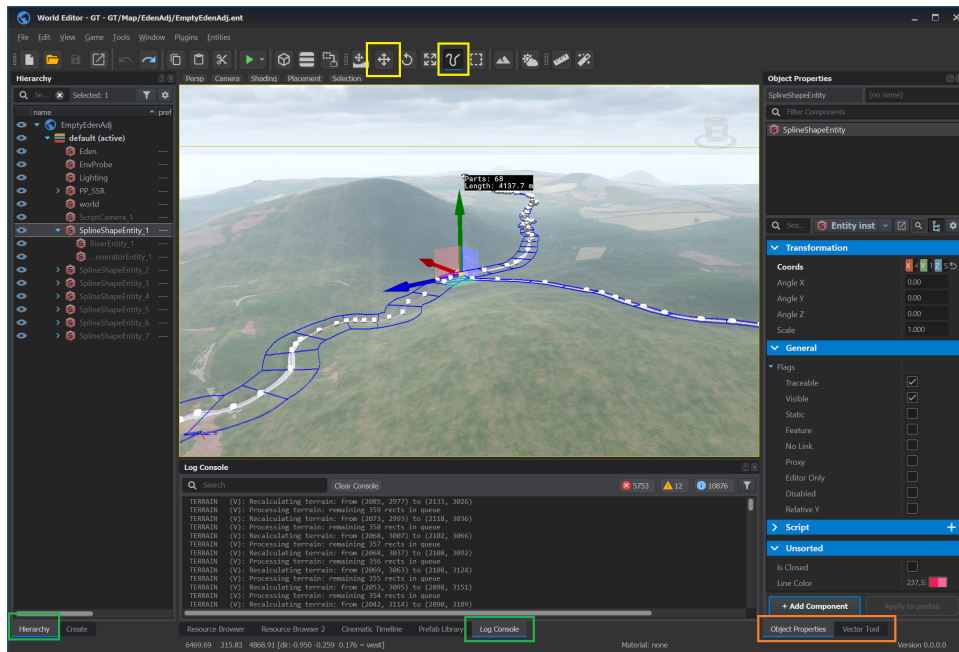


Figure A.3: The World Editor window after loading a world. Highlighted in green are the useful *Hierarchy* and *Log Console* windows. Highlighted in yellow are the *Move* and *Vector Entity* tools used to manipulate splines. Highlighted in orange are the tabs that switch between *Object properties* window and the current tool.

refer to non-essential missing data (since the attached build only contains data necessary for testing the tool).

6. All maps (except for *CustomMap*) already come with some premade rivers, which are, however, not applied to the terrain. These rivers can be found in the *Hierarchy* window. Every river is composed of a *SplineShapeEntity* object defining its shape and the per-segment parameters, and two child objects, *SCR_RiverGeneratorEntity* (which generates the terrain) and *RiverEntity* (which creates the river surface mesh). For information on how to apply these rivers to the terrain, see Section A.6.

A.2 Adding a new river

Adding a new river requires the user to first add a new *SplineShapeEntity*. To do this, select the *Vector Entity* tool, click the tool's tab in the bottom right corner of the main window, and click *New Spline* at the top of the window. Then, click anywhere in the main content area to place a spline origin, which will serve as the river's source. Then, right-click on the spline in the *Hierarchy* window and select the *Create SCR_RiverGeneratorEntity* option in the bottom part of the list. To add a river surface, right-click on the spline in the *Hierarchy* window and select *Create RiverEntity*. With the new *RiverEntity* selected, adjust the *Width*, *Spline Offset Up* and *Reverse Flow* options in the *Object Properties* tab to modify the river surface.

A.3 Generating the rivercourse

Select the *SCR_RiverGeneratorEntity* and click the *Object Properties* tab in the bottom right corner of the main window. All options related to shape generation are located in the equally named section (see Figure A.4).

Shape Generation Start Point determines how many points (excluding the initial one) of the existing spline should be retained. Setting a zero will cause the entire river to be regenerated. After

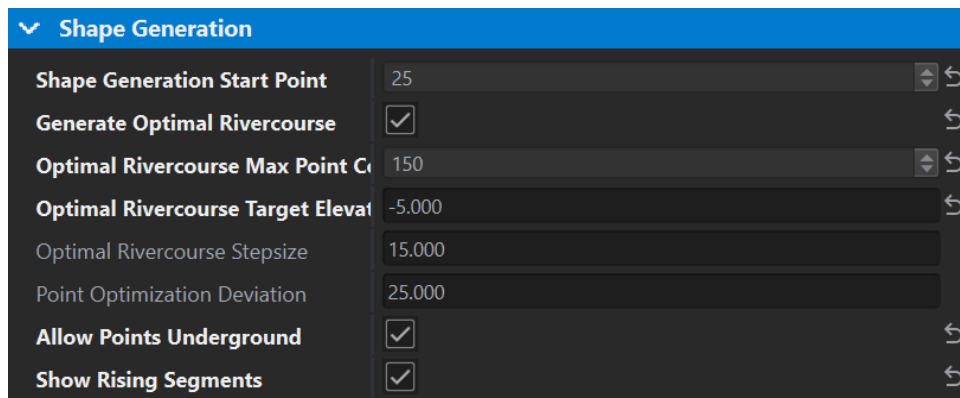


Figure A.4: The shape generation options.

entering a number, a hint with the text RG will be visible in the content area showing which point is selected. It is also possible to set the Start Point to a value out of the bounds of the current spline, in which case the actual start point will be limited to the current bounds. This is especially useful when generating the river in multiple shorter segments, as one can set it to an extremely high value and keep adding new points past the existing spline without adjusting the Start Point property.

Generate Optimal Rivercourse will start the shape generation upon enabling. If a large river is being generated, this can cause the Workbench to stutter momentarily (since Workbench scripts are executed on the program's main thread).

Optimal Rivercourse Max Point Count determines the maximum limit on the amount of generated points. This includes any points that will be considered unnecessary for the shape and optimized away. After reaching this amount, the generation will stop, and the user can change the *Shape Generation Start Point* property to continue adding new points past the existing spline.

Optimal Rivercourse Target Elevation determines the elevation at which the shape generation should stop.

Optimal Rivercourse Step Size determines the expected distance between two spline control points. Smaller values (5-15 meters)

are suitable for mountainous streams, and larger values (15-30 meters) are suitable for larger rivers flowing through flat terrains.

Point Optimization Deviation determines the minimal offset of a point from the line passing between its neighbors necessary for the point not to be optimized away. Points that do not meet this criterion are removed to reduce the memory footprint of the spline. Larger values are more suitable for larger rivers. Setting this value to 0 will disable any point optimization.

Allow Points Underground allows the generator to continue generating a river even if the newly added points will be deep underground. This option should only be enabled if the last added point is already underground (and creating a canyon), as it can cause the generator to continue adding points that will result in an unrealistic river.

Show Rising Segments highlights the points which would cause the river to flow upstream with a red hint visible in the content area.

After generating the points, the user can select the *Vector Entity* tool and select and move the points, add new points at the end of the spline (using the Ctrl+2xClick shortcut) or remove points (by pressing the Delete key). It is also possible to adjust a point's tangent by selecting the point and then clicking its' tangent, visualized by a blue line. When moving a point, it is recommended to switch the gizmo to the world space using the *World Space* button (see Figure A.2) to prevent moving the points along the Y axis unless intended.

A.4 Generating the parameters

Options for parameters generation are located in the equally named section (see Figure A.5).

Generate Parameters will start the parameters generation upon enabling.

Parameters Generation	
Generate Parameters	☒
Params Generation Start Point	0
Params Generation End Point	10000
Initial Water Discharge	1.000
End Water Discharge	16.000
Discharge Curve	
Braids Probability	
River Width Multiplier	0.750
River Depth Multiplier	3.000

Figure A.5: The parameters generation options.

Params Generation Start, End Point determine which points will be the parameters generated for. For instance, if, at some point, two rivers merge, it is possible to create parameters first for the section of the river before the confluence using a smaller discharge value and then using a larger discharge value after the confluence. It is possible to use values out of the bounds of the current river, in which case the indices will be clamped to the river's bounds.

Initial, End Water Discharge determine the water discharges at the actual Start and End Points of the parameters generation. To adjust the curve, double-click in the graph to add new points, drag points to move them, and press the Delete key to remove them.

Discharge Curve determines the evolution of the discharge between the Start and End Points.

Braids Probability determines the chance of multiple channels forming over the span of the river between the Start and End Points, as long as the sedimentation is sufficient and the flow speed and slope are low. Thus, setting the value to 1 does not guarantee that braids will form, but setting the value to 0 guarantees they will not.

River Depth, Width Multiplier allow the user to adjust the dimensions of the river by a set multiplier. Since rivers in video games are usually deeper but narrower than their real-world counterparts, it is recommended to use a depth multiplier larger than one and the width multiplier smaller than one (in my testing, 3 and 0.75 seemed to provide the best results).

A.5 Setting river parameters manually

Manually adjusting the river parameters is also possible. Select the *Vector Entity* tool and select the tool tab in the bottom right corner. Then, select a point of the spline that should be adjusted. A list of *PointData* objects assigned to the point will be presented in the lower half of the tool window (see Figure A.6).

If a point already contains a *RiverGeneratorPointDataFixed* object, the available parameters will be visible. Otherwise, it is necessary to add the data object by clicking on the + button and then selecting the appropriate entry in the menu¹. Dimension randomness and meander data can be added to a point by clicking the *Set class* button next to the appropriate entry in the properties list.

Section 2.3 provides an overview of the available options.

A.6 Applying the river generators

After creating the river shape and setting the river parameters, the user can carve the river into the terrain. This is done using the options in the *Terrain Adjustment* section in the *SCR_RiverGeneratorEntity's Object Properties* window (see Figure A.7).

Enable Heightmap Adjust determines if the river generator should be applied to the terrain. If the option is enabled, the generator will also react to any changes in the terrain done by other generators and the terrain sculpting tools. Generating larger rivers can take up to a minute - if the user launched the program with the

1. Objects *IRiverGeneratorPointData* and *RiverGeneratorPointDataInterp* do not give the user the option to change these parameters, and are only visible in the menu due to a technical limitation of the engine.

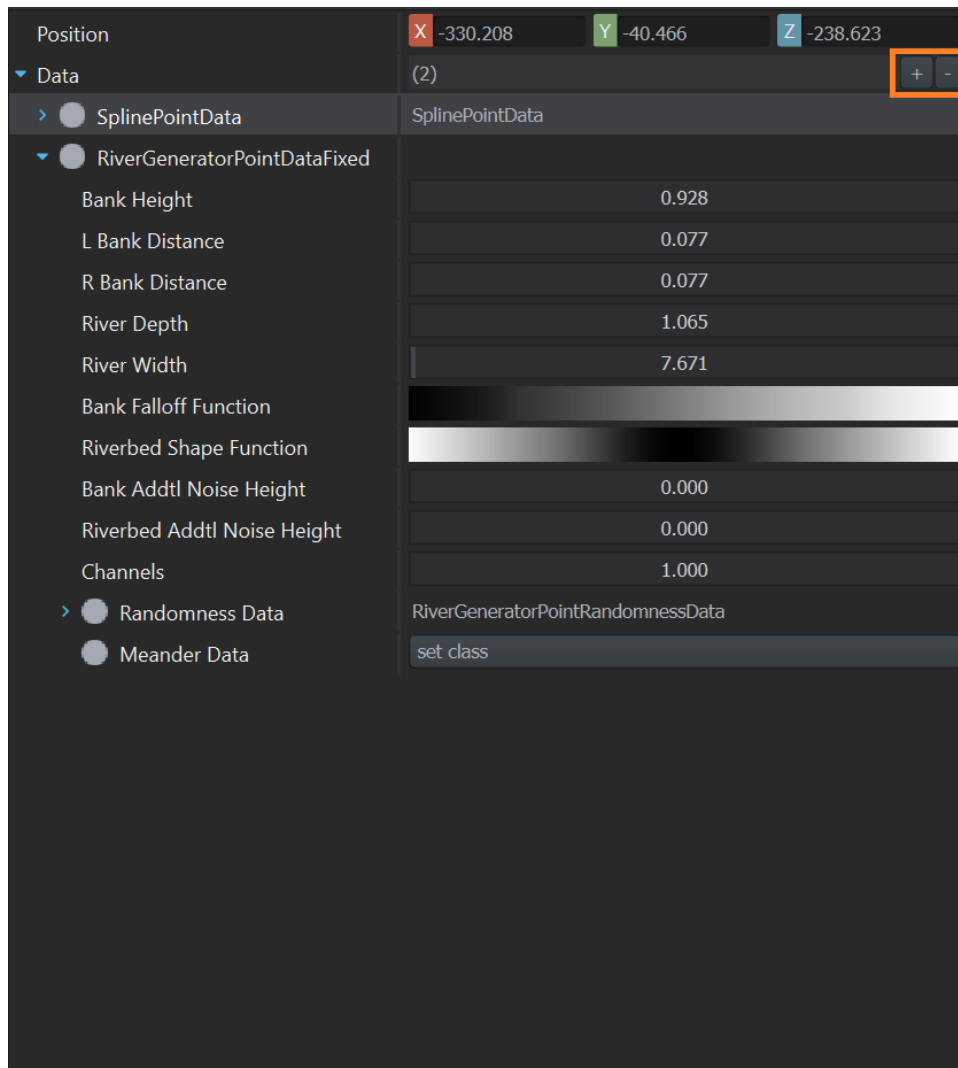


Figure A.6: The parameters available at each river point as presented in the *Vector Entity* tool window. Highlighted in orange are the + and - buttons used for adding or removing *PointData* objects.

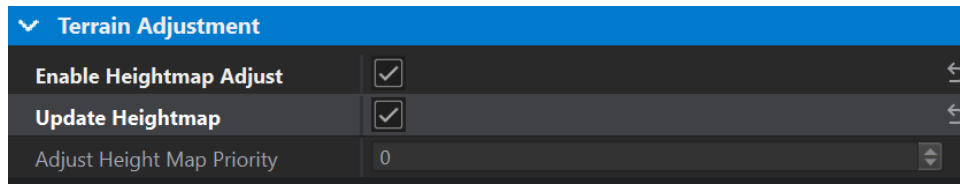


Figure A.7: The options related to applying the river generator to the terrain.

`logLevel -verbose` parameter, the number of terrain patches yet to be regenerated is reported in the *Log Console* tab.

Update Heightmap determines if the river should automatically regenerate sections affected by changes to the spline or the parameters. Disabling the option allows the user to adjust multiple values at once without needing to wait for the terrain generation to occur after every change.

Adjust Heightmap Priority allows the user to set the order in which the generators are evaluated, which is helpful if multiple types of generators are overlapping. As the rivers are currently the only type of available generators, and their riverbeds are blended based on their depth instead of the priority, this property can be ignored.

After opening any of the maps available in the build provided in the thesis archive, all river generators present on the map will be disabled. While the user can enable generators separately, this will lead to parts of the heightmap being evaluated multiple times needlessly. To enable all generators at once, the user can select them in the *Hierarchy* window (for instance, by entering the string "RiverGenerator" into the search bar at the top of the window and selecting all the results using the `Shift+Click` method), and only then check the *Enable Heightmap Adjust* option. This will enable all selected river generators at once. The same method can also be used for enabling or disabling the *Update Heightmap* option.

A.7 Creating a custom map

While the *EmptyEden*, *EmptyEdenAdj*, *Hill*, and *Mountain* maps cannot be manually edited, as, due to their licensing, they are compressed and read-only, the user can use the *CustomMap* map to create a custom terrain. After opening the world in the World Editor, click the *Terrain tool* button (or use the Ctrl+T shortcut, see Figure A.2 or A.8) in the toolbar. Then, select the *Terrain Tool* tab in the bottom right corner and then the *Sculpt* tab at the top of the tool window. From here, use the *Sculpt*, *Flatten*, *Smooth*, and *Noise* brushes. While using a brush in the content area, it is possible to use modifier keys Ctrl to adjust the radius, Shift to adjust the strength, and, in the case of the *Sculpt* tool, Alt to invert the effect.

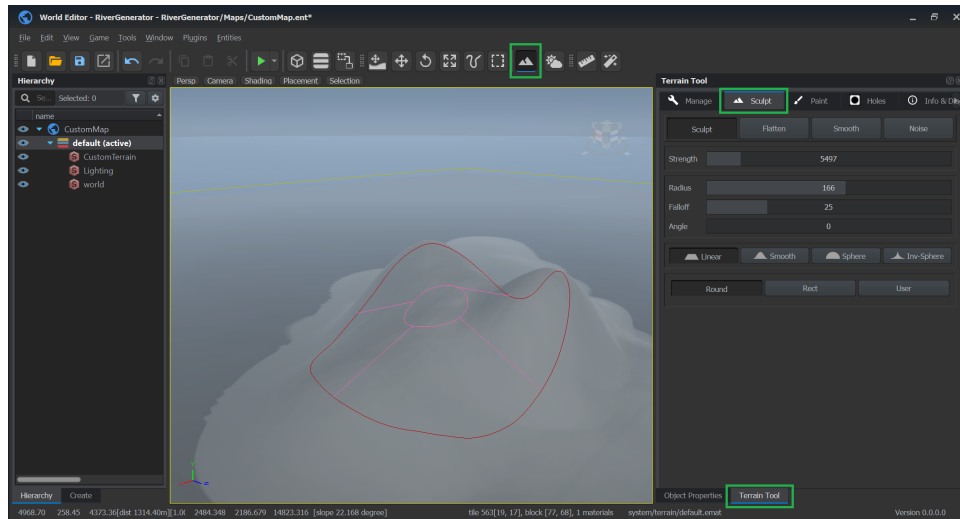


Figure A.8: Editing the terrain with the terrain tool on the custom map, with navigation to the sculpting tools highlighted in green.

B List of third-party code

The tool was implemented in the Enfusion Engine developed by Bohemia Interactive and uses various features already present in the engine, such as the spline evaluation and manipulation tools, scripting engine, river surface meshes generation, and terrain heightmap implementation. The core parts of the tool implementation use logic inspired by an implementation of the `RoadGeneratorEntity` tool included in the engine. A list of third-party code used by the Enfusion Engine can be found in the archive in the `workbench.zip/CoreData/licences` folder.

In addition, this thesis uses the following code:

- 1D Simplex Noise implementation initially created by Stefan Gustavson [38].
- Intersection of two lines in a 3D space implementation initially created by Ronald Goldman [39, p. 304].

C Electronic attachments

The electronic thesis archive contains the following files:

thesis.pdf - the thesis itself.

workbench.zip - a Workbench build for the evaluation of the river generator tool.

code_samples.zip - the most important parts of the code used in the implementation.

screenshots.zip - a few screenshots of the generated rivers (some, but not all, of which are also present in the Appendix D).

showcase.zip - a video showcasing the tool's options.

D Screenshots

The complete collection of 24 screenshots of the three rivers is available in the thesis archive in the `screenshots.zip` file.

- River *Everon A* (Figures D.1 and D.2) is a 1 kilometer long river created for the showcase video on the Reforger's Everon map. The shape of the river was generated with the few last segments (shown on Figure D.2) adjusted. The parameters were generated and then slightly adjusted in some areas to reduce heightmap resolution issues.
- River *Everon B* (Figures D.3 and D.4) is a 2 kilometers long river with a fully generated shape. The parameters were also generated and then again adjusted in some areas where the heightmap resolution was not sufficient.
- River *Hill* (Figures D.5 through D.10) is an 8 kilometers long river with a fully generated shape and fully generated parameters (except for the addition of the meanders parameter). Last section also shows both bank meanders and multiple channels.

Generating each river required the author to spend approximately 5-10 minutes setting up the generator, carving the river (which, in itself, takes seconds at most), making minor adjustments to the parameters and manually sculpting the surrounding terrain in places where it was lower than the surface of the river (which would cause the river surface mesh to be visible). Rocks, bushes, and trees, as well as the terrain materials, were added manually. All used 3D and 2D assets are part of the Arma Reforger game and thus property of Bohemia Interactive.



Figure D.1



Figure D.2



Figure D.3



Figure D.4

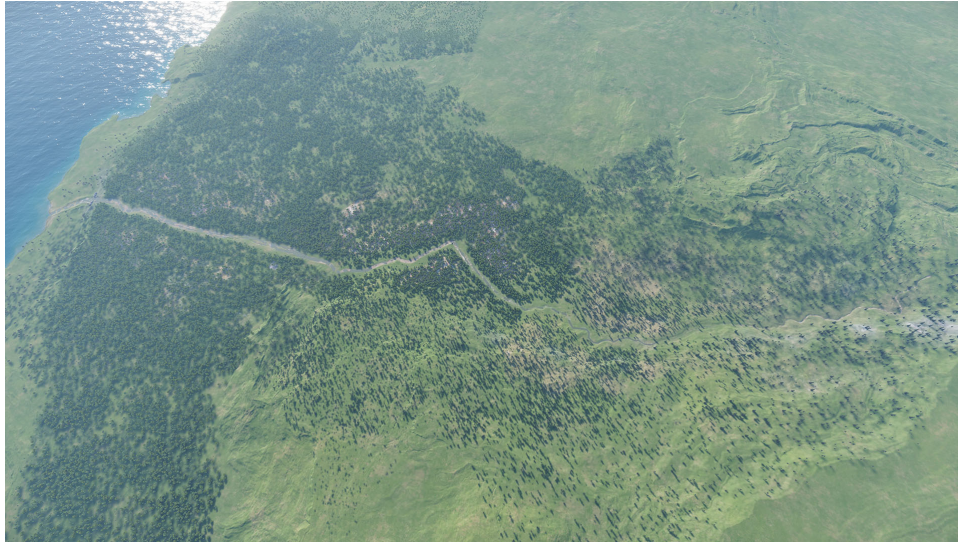


Figure D.5



Figure D.6



Figure D.7



Figure D.8



Figure D.9



Figure D.10