Procedural Generation of Landscapes with Water Bodies Using Artificial Drainage Basins

Roland Fischer · Judith Boeckers · Gabriel Zachmann

Abstract With the continuing trend towards larger and more detailed 3D worlds in various fields and industries, procedural generation is becoming increasingly important. An aspect that is still lacking research, however, is the procedural creation of landscapes that realistically integrate water bodies. Most previous work either encompasses extensive manual authoring, employs costly simulation-based approaches that offer little control, or produces artificially looking results. We propose a method for procedural generation of huge landscapes that focuses on creating realistically-looking river networks and lakes as well as a natural-looking integration. We achieve this by an approach inverse to the usual way: We first generate rivers and lakes based on artificial drainage basins and then create the actual terrain by "growing" it, starting at the water bodies. Our pipeline approach not only enables quick iterations and direct visualization of intermediate results but also balances user control and automation. That means, the first stages provide great control over the layout of the landscape while the later stages take care of the details with a high degree of automation. Our evaluation shows that vast landscapes can be created in under half a minute. Also, with our system, it is quite easy to create landscapes closely resembling real-world examples, highlighting its capability to create realistic-looking landscapes. Our implementation is easy to extend, highly compatible with external applications thanks to using heightmaps as underlying data structures, and thus, can be integrated smoothly into existing workflows.

This technical report is a much extended version of our paper at CGI [8], which is available online at https://doi.org/.

Keywords Procedural Generation · Water Bodies · Procedural Rivers · Terrain Generation · Drainage Basins

Roland Fischer
University of Bremen, Germany
E-mail: r.fischer@uni-bremen.de

Judith Boeckers
University of Bremen, Germany

Gabriel Zachmann
University of Bremen, Germany
E-mail: zach@uni-bremen.de

1 Introduction

Procedural generation of 3D landscapes is a research topic of great relevance, as the interest in large, realistic digital landscapes is steadily rising throughout many fields and industries. Prominent examples where such digital landscapes are employed are computer games, especially with the growing trend of so-called open-world games, but also movies, simulators, and virtual testbeds. An old but still relevant challenge, however, is to produce realistic and detailed terrains while keeping the workload in check. With greater and denser worlds the challenge is only getting more exacerbated. Numerous works have been presented on the automatic generation of landscapes and terrains using procedural generation techniques. The goals and approaches of the proposed works vary widely, some put the focus on highly realistic terrains and take a computationally expensive simulation-based approach mimicking natural processes such as plate tectonics and erosion, while others employ simpler and faster methods to generate plausible and more varied results. For instance, noise methods were popular, especially in earlier works, but today too. Naturally, there has always to be a trade-off between control and automation as fully procedu-

ral generated landscapes usually do not meet specific requirements but designing everything by hand is not viable either.

A sub-topic that got much less attention despite being highly relevant for large landscapes is the procedural generation and plausible integration of water bodies, i.e., mainly rivers and lakes but also other features such as smaller streams and ponds. While rivers/river networks, their natural processes, and interaction with the surrounding terrain have been – and still are – extensively studied in related fields such as geology, ecology, and hydrology [3,4,20], relatively few works focused on procedural generation of 3D representations of them in near real-time speed. Existing scientific models and simulations usually employ only 2D representations, are more focused on analyzing existing landscapes than creating novel ones, or are very time-consuming to perform. For instance, one popular model to create river networks is optimal channel networks (OCNs) [1,19]. Brown et al. [2] give a good overview of the different works and approaches to create digital rivers throughout the various research fields.

We propose a method and pipeline for quick and easy procedural generation of large, plausible-looking landscapes which include and integrate believable water bodies, i.e. river networks. In our approach, we mimic the mutual influence between terrain and water bodies by first generating the rivers and lakes based on artificial drainage basins, and then computing the final terrain. This way, we get more natural-looking landscapes, than by retroactively adding rivers to a terrain. In order to demonstrate our proposed approach, we have developed a prototype application in Unity. In this prototype, we have applied a pipeline approach that makes it easy to evaluate intermediate results and emphasizes a workflow with quick iterations. Finally, we have conducted an extensive evaluation of our proposed system.

2 Related Work

One of the oldest approaches for procedural terrain generation is to use subdivision techniques such as the midpoint displacement and the diamond square algorithms, and noise functions (e.g., simplex and ridge noise), as they are able to produce fractal-like structures, which are also often found in nature. Moreover, such techniques are, generally, relatively easy to use, highly scalable, and computed quickly. A comprehensive overview of various noise functions is given by Lagae et al. [16]. The drawbacks of those techniques are the intrinsic lack of control over global features, and the un-intuitive parameters, which make it hard to create geologically plausible landscapes.

A popular approach to providing intuitive control to the user is to add an authoring phase at the beginning, most often in form of a user sketch that acts as a high-level constraint for the subsequent terrain generation [10,22].

An approach to create more realistic terrains is to mimic or simulate natural processes. For instance, Michel et al. [18] create folded terrains with mountain ranges by using simplified plate tectonics that is based on user sketches and Cortial et al. [6] approximate the movement and collision of user-authored tectonic plates on a planetary scale. Fischer et al. [9] combined noise-based methods, a simplified climate simulation, and digital elevation model (DEM) examples to create large landscapes with plausible biome distributions. Simulation-based techniques most often focus on thermal or hydraulic erosion, the latter normally encompassing some form of fluid simulation. One example is Mei et al. [17], who used an adapted shallow-water model to calculate the erosion and deposition process as well as the sediment transport. The proposed work was implemented on the GPU. Stava et al. [21] further improved the method and combined two hydraulic erosion algorithms. Cordonnier et al. [5] combined tectonic uplift from user-provided input maps and simulation of hydraulic/fluvial erosion based on the stream power equation to generate plausible large landscapes. However, despite efforts to speed up the computations, most simulation-based approaches are very time-consuming, especially if applied on large-scale terrain. Another disadvantage is the lack of intuitive control over the generated terrains.

Realistic-looking large-scale terrains also can be created using example-based procedural generation techniques such as texture synthesis. For instance, Zhou et al. [25] employ patch-based texture synthesis to generate terrains based on user sketches and example DEMs. Gain et al. [11] instead switched to parallel pixel-based terrain synthesis for higher efficiency; user control is provided by several modifiable, local constraints. Guerin et al. [14] presented an example-based authoring pipeline in which the user provides a quick sketch of the main terrain features, and then a set of neural-network-based terrain synthesizers creates the corresponding terrain. The synthesizers – which are Conditional Generative Adversarial Networks – get trained on real-world example data. Naturally, example-based methods are limited by the available example data and can only replicate terrain features and landforms that are represented in the input DEM. Also, high-level geological constraints and the correct relations between large-scale features such as drainage basins are usually not taken into account.

Relatively few works explicitly focus on procedural rivers and water bodies as initial terrain-defining elements, although river networks play an important role in the natural formation of the terrain. Kelly et al. [15] were the first to propose the idea of procedurally generating terrain based on river networks and corresponding drainage basins. Here, the river networks were generated based on constrained midpoint displacement, and then the terrain was computed accordingly. Derzapf et al. [7] employed a similar approach but applied it on a planetary scale. In the work by Teoh [23], the terrain generation starts, too, by first procedurally creating river networks. In this case, rivers are grown from randomly placed outlets around the land region. In contrast to these works, Genevaux et al. [13] explicitly take hydrological knowledge into account, additionally, initial user sketches provide more control. Based on the sketch, a river-network graph is created, river segments get classified into different types of watercourses, and the surrounding terrain gets computed using a hierarchical terrain construction tree. Zhang et al. [24] present a similar approach but generate the rivers based on Tokunaga river networks and calculate the surrounding terrain using a diffusion process.

For a more comprehensive overview and discussion of procedural terrain generation techniques, we refer to the recently presented work by Galin et al. [12].

3 Overview of our Approach

In this section, we will present an overview of our proposed methods and pipeline. First of all, we will briefly discuss different approaches to procedural terrain generation that include and emphasize river systems, and explain the reasoning behind our approach.

The first group to consider are purely simulation-based approaches. These are able to produce realistically-looking terrains with river networks, e.g., using erosion simulation. As we prioritize a quick and easy generation over absolute realism, though, we have decided against these simulation-based approaches. Another approach that follows the classical order of first generating the terrain and then adding rivers to it is to use pathfinding algorithms. For instance, an adapted A* pathfinding algorithm can be used to follow the terrain downwards from randomly placed sources. This not only has the advantage of being computed rather quickly but also that lakes can be easily computed by just defining the searched area as a lake, which tends to be in local minima. However, in our experience, the river networks and their integration into the terrain were not convincing, as they did not respect geomorphological constraints. In this paper, we instead propose to follow the more natural "rivers first" approach, specifically, first generating river networks based on artificial drainage basins and then modeling the final terrain after them. Accordingly, we will present several methods and an integrated pipeline to allow for that.

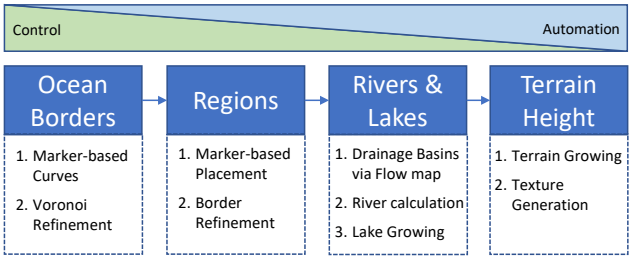

Fig. 1 Overview of our procedural generation pipeline. The first stages feature more user control, the latter ones provide more automation.

For the implementation of our pipeline, we have used heightmaps as data structures for data and terrain representation, as they have a smaller memory footprint and, most importantly, provide much greater compatibility with external applications than voxels. Even though we do not aim for real-time generation (i.e., an online algorithm), it is very important to facilitate a quick workflow for the users, which means having fast computation times as well as direct visualization of intermediate results that can be made modifications upon. These considerations lead us to employ a pipeline approach in which each step should be computed in a matter of seconds, be repeatable if modifications are desired, and the results directly be applied on a proxy mesh for inspection. Fig. 1 depicts a high-level overview of our approach. We start with the general landscape layout by letting the user author the landmasses using marker-based curves. Then, different regions can be marked (e.g. flatland, or mountains). Following this, the river networks, including lakes, are computed based on artificial drainage basins. Finally, the terrain height gets computed based on the previous steps. In the earlier stages, we emphasize providing the user with more control over the algorithm, while the later stages have a higher degree of automation. The reasoning for this is that the user should have a great influence over the general layout and shape of the landscape and its landmasses, which are defined in the earlier stages of the pipeline, but not be overwhelmed with a host of detail decisions all over the landscape. Smaller modifications at the places where it is deemed necessary could be better done afterward via a polishing pass with a sculpting tool.

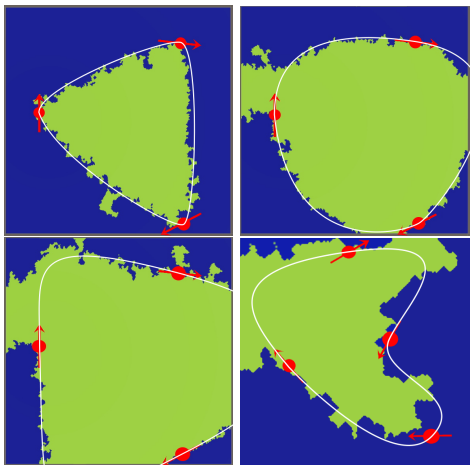

Fig. 2 Authoring of the general shape of the landmass borders using marker objects which act as spline control points (red points with arrows). The white curve represents the generated spline and the green area depicts the final refined landmass. First 3 images: increasing strength values (reducing the curvature of the tangent).

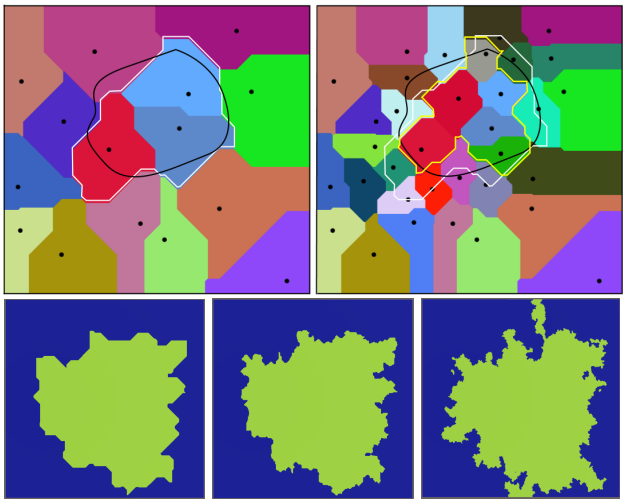

Fig. 3 Border refinement based on Voronoi diagrams. Top row: the two iterations of border refinement. The left image shows the result after the first iteration (the black ellipse depicts the initial spline, and the white outline the ocean border). The right image shows the second refinement iteration with the final border (yellow outline). Bottom row: The left image shows the land after one iteration, the middle image after two iterations, and the right image after two iterations with noise-based sampling.

4 Our Terrain Generation Pipeline in Detail

4.1 Ocean Borders

Our pipeline begins with the coastlines that separate the landmasses from the ocean. As the user should have great flexibility to shape the general terrain to his needs, we prioritized providing great manual control instead of excessive levels of automation for this part of the pipeline. To define the overall shape of a landmass, the user can place marker objects throughout the scene, which we then compute a closed curve from. This curve defines the coastline: the area within forms the landmass, and the rest represents the ocean (or the other way round, depending on a user setting), see Fig. 2. Multiple landmasses can be built by repeating the process. If no markers are placed, the whole scene forms a single landmass without an ocean. The markers contain location information, a rotation, and a strength parameter affecting the marker's influence, specifically, the curvature of the tangent (higher strength values resulting in less curvature). The curve is then computed by interpreting the markers as knots/control points of a cubic spline, the evaluation is done in the order the markers were placed. The final segmentation into land and ocean is stored in a regular grid whose granularity can be set by the user.

In order to improve the outlines and get a more natural look, we have developed multiple refinement options for constrained randomization based on Voronoi diagrams. Generally, we randomly place Voronoi sites on the map and define the regions using the Manhattan distance. We use this metric, as the computations are done on a regular grid; other metrics can be applied, too, though. In the first refinement variant, each Voronoi region gets classified to represent landmass or ocean, depending on if the corresponding Voronoi site is inside or outside the spline, see the top-left image in Fig. 3. This ensures that the general layout defined by the spline is retained, but the actual border is randomized. The granularity of the Voronoi diagram can be set by the user. To efficiently produce finer, more detailed borders, this refinement step can be followed up by a second iteration in which additional Voronoi sites are placed around the previously computed border, see the top-right image in Fig. 3. Alternatively, this second iteration can be applied using a priority queue based on (fractal Perlin) noise for the growth of the regions, see Algorithm 1 for details of this process. Although computationally more expensive, it produces more varied results through more inhomogeneous region growth and can lead to the formation of small islands. The noise parameters influence the output, e.g., a higher frequency produces finer structures, and more octaves enable features of different scales. Fig. 3 (bottom row) shows an island with the different refinement variants applied.

Algorithm 1 Border refinement (2. iteration, Voronoi + noise)

Require: 1. refinement iteration done, $RegionMap$ set
  $B \leftarrow BorderCells$ ▷ From 1. iteration
  $Q \leftarrow []$ ▷ Empty priority queue
  for all $VoronoiSites$ **do**
    $P \leftarrow random(B) + randomXY(MaxRadius)$
    $Q.Enqueue(P, Noise(P))$
  $B \leftarrow []$
  while Q not empty **do**
    $C \leftarrow Q.Dequeue()$
    for all N **do** ▷ Valid, unsearched neighbor cells
      $RegionMap[N] \leftarrow RegionMap[C]$
      $Searched[N] \leftarrow true$
      $Q.Enqueue(N, Noise(N))$
    if $CheckIfBorder(C)$ **then**
      $B.Add(C)$

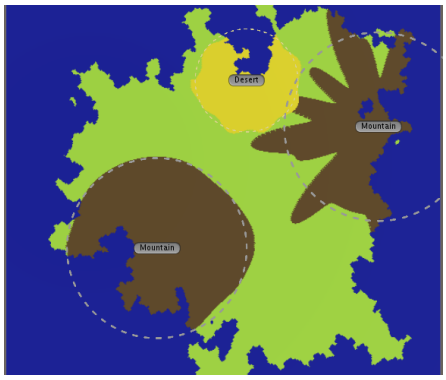

Fig. 4 Segmentation of the land into different regions (green: flatland, yellow: desert, brown: mountains). Region borders can be customized using noise (note the different characteristics of the borders).

4.2 Regions

After the landmasses are defined, the next step is to partition them into regions of different terrain types. Terrain types we have implemented are flatland (default), mountain, and desert. Others can be added easily, though. One option to divide the landmass into different regions could have been to use cellular noise. However, in this pipeline step, we again focused on giving the user great control over the layout of the regions by allowing the manual distribution of regions over the terrain. Alternatively, a random distribution is available too. A region is defined similarly to the coastlines in the previous step by placing region markers. This time they contain parameters regarding the terrain type, the extent, and its border, which is randomized using noise. Fig. 4 illustrated an example terrain with three regions and different border characteristics. Region overlaps are solved according to the placement order. After the regions are placed, we iterate through all cells that represent land and store the id of the region they lie in.

4.3 River Networks and Lakes

The next step in the pipeline is the generation of the river network. This is one of the most important steps in the whole pipeline, as it directly impacts the eventual terrain/heightmap generation. When analyzing naturally generated terrain, the landscape can be divided into catchment areas – also called drainage basins. These are the areas where water is collected by surface runoff (e.g. from precipitation). They are often divided by mountains or hills. This means that by artificially generating those catchment areas, we know afterward where we can place hills and mountains. To generate the catchment areas, we have developed a method inspired by optimal channel networks [19,1]. Similarly to them, we define a finite graph $G(V, E)$ over a regular grid spanning the landscape. The nodes $v \in V$ correspond to the cells of the grid, and the edges $e \in E$ link neighboring nodes and enable the flow of water between them. We then construct a spanning forest F over G that acts as a flow graph/flow network, i.e., G gets partitioned into a number of spanning trees T – acyclic, directed, rooted sub-graphs. Each outlet acts as a root of one of these trees, which each represent one river network or its catchment area. Our procedure is shown in the image sequence of Fig. 5 and starts by placing down several potential outlets around the coasts (left image) and also around mountain regions. The separation between those outlets happens so that the rivers can be generated differently based on terrain type. The number of outlets can be set by the user separately for each region type. Placing outlets around desert regions is excluded from this process. Then, we compute the catchment areas by construction of the spanning forest that indicates the flow directions. This is done by calculating random flow directions for all cells in the uniform grid and storing them in a flow map (middle image) that procedurally connects all cells from mountain and flatland regions. This process starts at the river outlets by adding all outlet cells into a fringe set. Then, consecutively, random cells from this set get selected and their unsearched, valid neighbor cells get processed by assigning the flow direction (pointing to the current cell) and, in turn, adding them to the fringe set. By using a random selection order, we guarantee a homogeneous growth of the spanning trees/propagation of the flow map and, thus, the drainage basins. This process also avoids desert regions to ensure that they stay dry when placing the rivers. The number of outlets influences the shape of the river networks. The fewer outlets are gen-

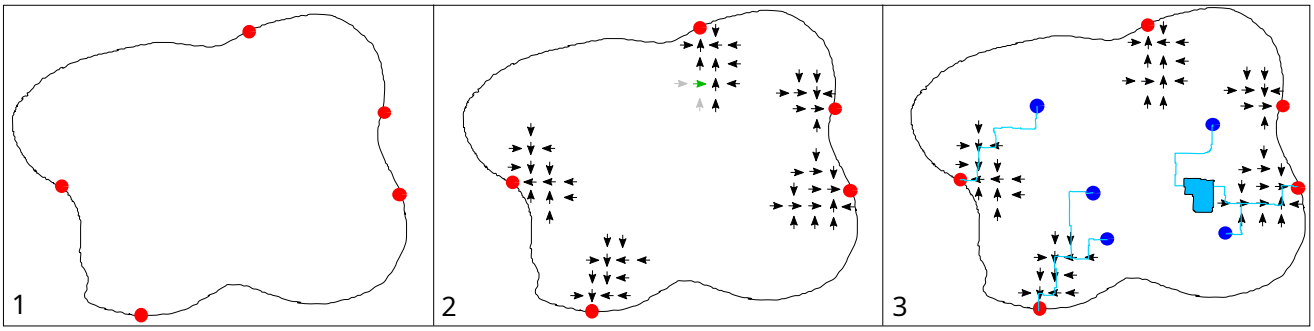

Fig. 5 The process of computing rivers and lakes. First (1), random outlets are selected at the ocean borders (red dots). Then (2), we compute the flow directions across the land (see arrows), starting at the outlets and then consecutively and randomly selecting previously evaluated cells (green arrow in the highlighted area) to process their unprocessed neighbors (grey arrows). Eventually (3), we randomly place river sources (blue dots) and create the actual rivers by following the flow map (blue lines). Lakes are grown from random points on the river.

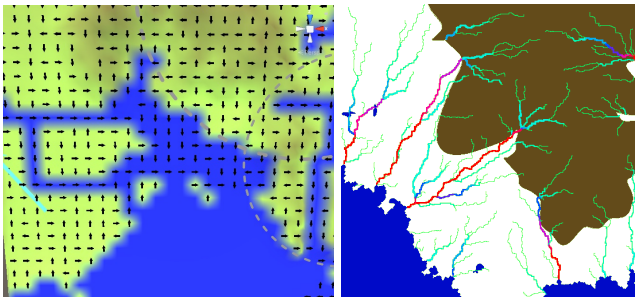

Fig. 6 Left: An example map with arrows roughly indicating the corresponding flow map. Right: A map highlighting the river strength/width with different colors (green indicating weak rivers and red strong ones).

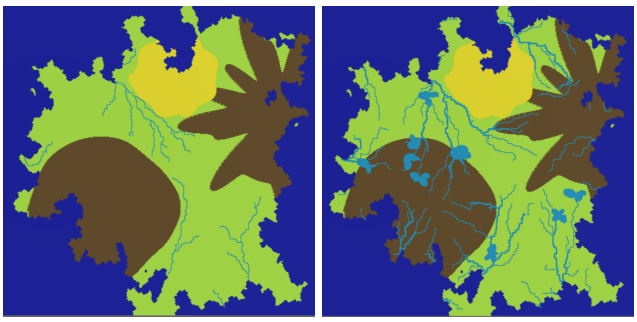

Fig. 7 Left: A terrain with rivers generated in flatland (green). Right: The same terrain as in the left picture but with rivers of varying width and the addition of lakes.

erated compared to the number of river sources, the higher the branching factor in the final river networks will be. This is because the individual rivers, starting at their sources, are more often routed to the same outlet and, thus, join somewhere along the way. Important to note is, however, that not all outlets will necessarily have rivers run into them. In Fig. 6 (left), an example landscape is shown with arrows depicting the corresponding flow map (due to the perspective there are slight offsets).

Following these preparations, we can proceed to generate the actual rivers. A possible approach for this would be to calculate the amount of water that would flow through each cell. This could be done by placing water evenly on the grid and following along each water unit with the previously generated flow directions while adding up how many water units traverse each cell. Rivers could then be placed in every cell that has an amount of water higher than a specified threshold. This method would work but lead to a relatively even distribution of river sources. Instead, we place the river sources randomly on the map and then follow the flow directions until the ocean is reached (right image of the image sequence of Fig. 5). This yields the advantage of a more random distribution of sources. The amount of river sources placed is again controlled by the user and can be set separately for mountain and flatland regions. An example terrain with multiple rivers is illustrated in Fig. 7 (left). Users also can set the chance for a river being a dried-out riverbed instead of a normal river. In our implementation, each river adds a certain amount of strength to a cell when flowing through it, thus, when two rivers join, their respective strength is combined from thereon. Based on this strength, we calculate the width of the river and accordingly assign more cells to it. Fig. 6 (right) depicts a color-coded example of the various rivers and their respective strength (red: strong, green: weak). The calculation of the width can be skipped by a user setting, though. It is also possible to set a maximum width for rivers. This can be useful when using a calculation grid with a lower resolution where one cell already covers a larger area of the final map.

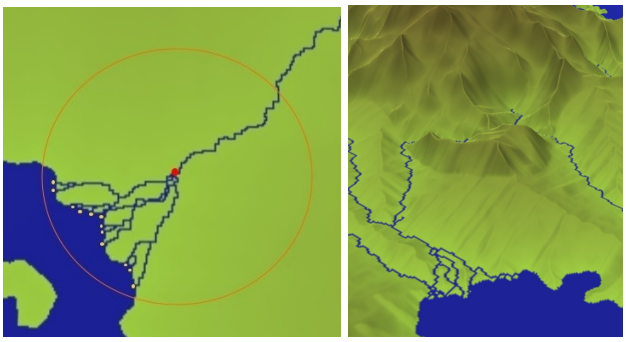

Fig. 8 Two example landscapes in which river deltas were computed (using different visualization modes). The red dot in the left image depicts the position from which the river delta computation was started, the circle shows the radius of the local flow map, and the yellow dots mark the stream outlets.

For each cell a river travels through, there is a chance to generate a lake. The size of the lake depends on the strength of the river cell it is generated from: the higher the strength, the more likely it becomes for the lake to be bigger. In addition, the size is limited by user-set maxima and minima. To produce lakes that vary in shape, a noise function is applied to the lake borders. The user can set the frequency, the strength, and the number of octaves of the noise function. All the cells inside the border are classified as lake. In order to avoid overfilling the map with too many lakes and to give control to the user, it is possible to set a maximum amount of lakes generated for mountain and flatland regions. In addition, the lake generation is limited to one lake per river source. After generating a lake, the river generation continues. Fig. 7 (right) shows an example terrain with lakes and river networks in which the rivers have different strengths.

With our approach to create river networks, we are also able to generate distinct river deltas. For this, we perform a second, locally-bound iteration of generating drainage basins and then rivers. However, here we reverse the process: If a river delta should be generated, we select a river cell instead of a coastline cell as the starting point from which we compute the flow map; in this case only a regional one. Then, we select multiple coastline cells to be outlets from where rivers – the eventual distributaries of the delta – are created by following the computed regional flow directions back to the original river cell. We allow this process to randomly occur in the general vicinity of the ocean, see Fig. 8 for examples of this process.

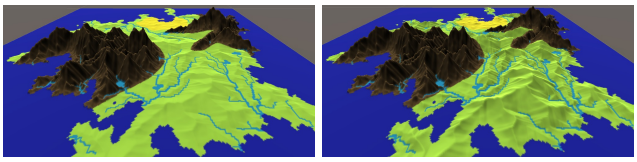

Fig. 9 Final terrains with computed height. Note that increasing the slope parameter for the flatland (green) leads to higher, more pronounced hills (right image).

4.4 Terrain

After generating the rivers and lakes, we calculate the terrain's height based on them and all the information that was produced in the previous steps of the pipeline. The terrain will be "grown" starting at the oceans, rivers, and lakes and continuously rise while departing from them. To do this, all starting cells are added to a priority queue with the priority being the initial height of the cells. The initial height of water cells was computed during the river generation in the previous stage: cells marked as river directly get assigned a height which steadily increases from the outlet onwards based on a region-based steepness, although a distance-based curve is possible too. When a cell is removed from the queue, all the neighbors that have not been traversed yet will receive a new height by the addition of a growth value. The growth value is calculated by taking into account the terrain growth factor of the region the cell is in, and a random value obtained from noise functions that also depend on the region. The height value is also interpolated between the different region types to allow for a smoother transition between them. The exact height calculation for a cell x' which is the neighbor of an already computed cell x is described by the following equation:

$$h(x') = h_x + s_t \cdot b_d + s_r \cdot (1 - b_d) + |n_t \cdot a_t| \quad (1)$$

In this, h_x is the height of cell x, s_t is the terrain/region-dependent growth factor which itself is computed using a distance-based interpolation between all regions in the area, b_d is a distance-based blending factor, s_r a growth factor for rivers, n_t the noise output whose frequency is again terrain-dependent, and a_t is a terrain-dependent amplitude. The latter is calculated as $a_t = s_t \cdot k_t$ in which k_t is a terrain-dependent noise strength. Fig. 9 shows example terrains with computed height; note the more pronounced hills in the right image. The growth value itself does not change the final height of the terrain, though, as it is difficult to know exactly what the result would be. To guarantee the final height is controllable, the heightmap is scaled to fit the global world width and height parameters that can be set by the user.

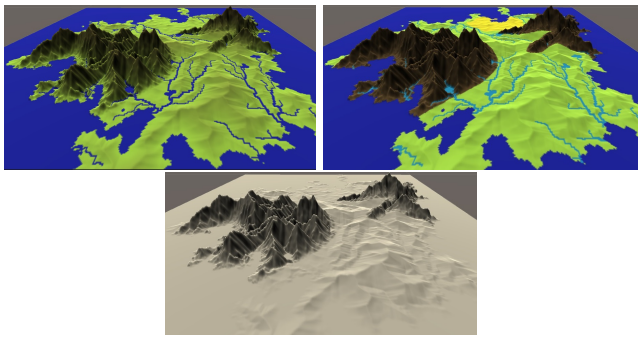

Fig. 10 Different rendering modes highlight different aspects of the terrain. On the top left is the normal-texture mode with water bodies depicted in blue and a height-based color gradient for land. In the top right is the region mode, in which each region gets a different color, and rivers and lakes get colored in light blue. The bottom image shows the height mode with a black and white normalized-height-based gradient.

4.5 Visualization

The last step of the pipeline is the visual representation of the generated terrain. This is not part of the actual terrain generation but serves an important role in giving the user information about the results of the previous steps which allows for quick changes if desired. To visualize the terrain, we generate a mesh by first assigning a vertex to each cell of the heightmap and then calculating the corresponding triangles and UV coordinates. In default mode, the vertices' height is directly taken from the heightmap, but we also support a mode in which the mesh omits the height and stays completely flat. This mode can be useful for earlier pipeline steps, e.g., the border generation, where the height information is not relevant yet, and a flat map is easier to evaluate. Finally, we generate textures for the terrain mesh. The user can choose from multiple rendering modes using different textures, which are illustrated in Fig. 10: The normal-texture mode displays a gradient that is dependent on the height, the region mode visualizes each region with a separate color, and the height mode displays a normalized heightmap as texture.

5 Evaluation

In order to evaluate our proposed procedural system, we present a brief complexity analysis, extensive practical performance measurements, and a qualitative evaluation.

5.1 Complexity Analysis

The overall run-time complexity of our pipeline is

$$O(n_c \cdot n_s \cdot n_l) \tag{2}$$

with n_c being the number of grid cells, n_s being the number of river sources, and n_l being the number of lakes. This means that the time complexity depends linearly on the number of river sources, the number of lakes, and the number of cells in the grid. Similarly, the space complexity is $O(n_c)$.

5.2 Performance Evaluation

After the theoretical considerations, we did extensive real-world performance measurements of our system as a whole as well as of each pipeline step individually. All performance measurements were done using a PC with Windows 10, an Intel i7 7800x processor, 16 GB of main memory, and an Nvidia GeForce 2070 graphics card. As the performance is mainly dependent on the number of grid cells n_c, we conducted all measurements with sizes of $n_c = 512^2, 1024^2, 2048^2$ and took the median of 20 runs.

In Fig. 11 we illustrate the computational time of the whole pipeline over the different grid sizes, whereby the timings of the individual pipeline steps are stacked on top of each other. As we can see, the computation

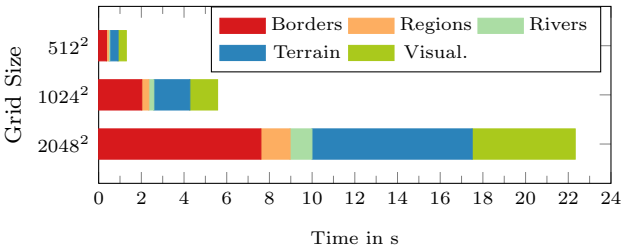

Fig. 11 The calculation times for the complete pipeline over multiple grid sizes. Even with a grid size of 2048^2 the whole pipeline gets computed in under 25 seconds.

is very fast: with low to medium-sized grids, the whole process is done in a couple of seconds, single steps being computed nearly in an instant, and even with the biggest tested grid resolution of 2048^2, the pipeline gets computed in under 25 seconds. Looking at the timings for the individual pipeline steps, the calculation of the region and rivers is the fastest and, compared to the other steps, negligible throughout all resolutions. At lower resolutions of 512^2, the calculation of the terrain's height and the visualization take the most time with 0.39 and 0.38 seconds, respectively. At higher resolutions of 2048^2, however, the border computation takes

the longest with 7.6 seconds, followed by the terrain with 7.5 seconds. The reason for this is that the computational time of most pipeline steps growths with factors closely around the expected one of 4 that corresponds to the linear growth regarding the number of grid cells (quadratic regarding grid side-length) which we established in the theoretical complexity analysis. The time for the border calculation, in practice, growth with factors around 5, though.

Investigating deeper what exactly causes the computational time in the individual steps, we find that in the river step the calculation of the drainage basins via the flow map takes up $> 88\%$ of the time while the following computation of rivers and lakes takes nearly no time, see 1. Accordingly, the number of rivers and lakes does not have a significant effect. The main factor regarding the performance is therefore the overall grid resolution. Some other parameters, however, also have a notable influence on the needed time for computation. For instance, in the first pipeline step – the border calculation –, the second iteration of border refinement takes considerably longer than the first one, as a higher number of Voronoi points is used. Also, if the second refinement iteration is applied with additional noise (default setting), it takes even more time to compute, as in this case, we use a priority queue. This is also the reason for the higher practical growth factor of this step. Regarding the visualization step, the computation of the textures takes roughly four-fifths of the time of the step while the mesh generation itself only takes one-fifth.

Table 1 Detailed timings of some pipeline steps for a grid resolution of 2048^2.

Step	Substep	Time in ms
Borders	Refine. (1i)	851
	Refine. (2i)	1601
	Refine. (2i+N)	7600
Rivers	Drainage B.	796
	Rivers+Lakes	100
Visual.	Mesh	800
	Textures	3426

5.3 Qualitative Evaluation

To our knowledge, there are no metrics to quantify the quality and realism of procedural terrains and water bodies. Thus, in order to evaluate the quality and plausibility of the terrain generated with the proposed system and to showcase its versatility, we did a qualitative evaluation. For this, we have created several landscapes with different settings, which can be seen in Fig. 12, and reviewed the production process as well as the results regarding usability, flexibility, and plausibility. With our system, it is possible to produce a vast variety of shapes for the coastline. Single continents, as well as island groups, can be created by varying the number and position of land markers. Generally, we found the process very efficient and flexible; simpler shapes can be realized very quickly with just a few markers, but by using a greater number, the user also can create more complex worlds. Similarly, the whole process to create a terrain is very easy and straightforward, as our pipeline design allows for quick iterations and the saving of intermediate results. Also, although we provide many parameters to fine-tune each step to the user's liking, in most cases the majority of them don't necessarily have to be changed and our pre-configured default settings will suffice.

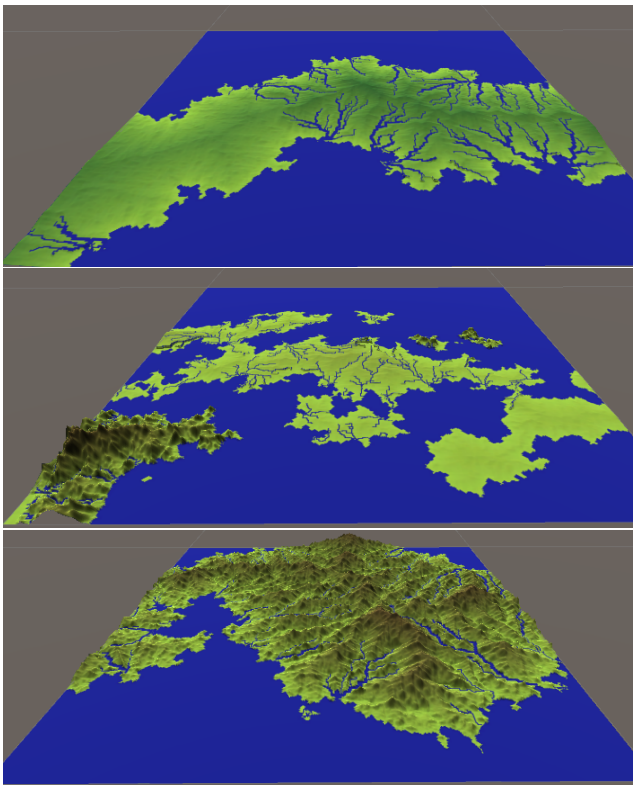

Fig. 12 Example landscapes generated with our system. Note the high variability and plausibility.

Furthermore, several different height profiles can be generated. It is possible to generate large-scale maps, such as the examples in Fig. 12 but also landscapes at smaller scales, as shown in figure Fig. 13. Thanks to our focus on water bodies and the approach to create river networks before the final terrain, the procedural

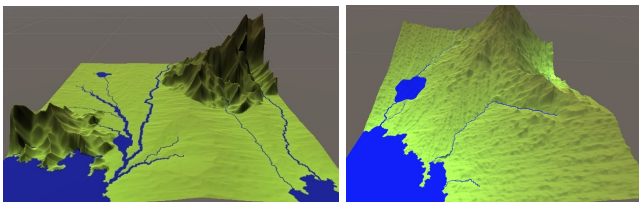

Fig. 13 Examples of small-scale landscapes.

landscapes produced with our system are quite natural looking and feature plausibly embedded river networks that recreate the typical dendric structures from the real ones. In general, we found that the generated results look very plausible and, presumably, the majority of different demands on the produced landscapes can be satisfied.

To further evaluate the plausibility of the generated terrains, we have compared them with parts of the real world's terrain based on publicly available height data. For this comparison, we took digital elevation models (DEMs) – which represent elevation data of the real world's terrain –, constructed 3D meshes of them, and attempted to replicate the real terrain as closely as possible with our system while only investing a reasonable amount of time (a couple of minutes). As an example, we randomly took a section of the Severo-Evensky District in Magadan Oblast in Russia (61.21703, 160.21836) as a real-world reference. Fig. 14 shows the comparison between the mesh representations of both landscapes, the left image shows the real terrain, and the right one our systems replication of it.

As can be seen, it is possible to recreate a similar general shape of the coastline. Because the generation is heavily based on random components, it is impossible to generate a coastline that matches exactly. The mountain ranges are distributed with a good approximation of reality. We were not able to acquire real-world references with satisfactory information about river networks, therefore, the map was generated only using dried riverbeds (no lakes, no explicitly visualized rivers). It is not possible to perfectly match the behavior where terrain touches the world border as the real map is a part of a larger landscape and thus, rivers flow through the border. Our terrain generation algorithm does not have information about the terrain outside the grid borders and thus cannot replicate this behavior. However, the inland parts of the river networks were generated in a believable way. Even though the pathways of the riverbeds differ from the original directions, the individual parts of river networks have similar overall shapes. This can be observed, for instance, in the northern parts of the mountains in Fig. 14. The heightmaps of both terrains are depicted in Fig. 15. Again, the general shape, as well as the dendric structures caused by the rivers, resemble the original, although they do not match perfectly. In general, it was possible to create a good approximation of the real terrain.

6 Conclusion and Future Work

With this work, we have presented a system for the procedural generation of vast landscapes with a focus on the natural and realistically-looking integration of water bodies. This is achieved by the approach of first generating river networks and lakes based on drainage basins and then the actual terrain. A quick and agile workflow is facilitated thanks to our pipeline design in which each stage is computed and visualized in a matter of seconds. According to our performance measurements, a high-fidelity landscape (grid resolution of 2048^2) can be computed in under 25 seconds. In order to balance the amount of control, usability, and efficiency, we have designed the first pipeline stages to allow the authoring of the general landscape and its layout, while the later stages are more automation-heavy on the terrain details. Of course, our methods in the various stages of our pipeline are easily modifiable to much more or even less control, so it can be easily adapted to different needs in different workflows. Our qualitative evaluation demonstrated the great variability of our approach and a dedicated comparison with real-world terrain based on DEM data illustrated the capability to quickly create terrains strongly resembling the real ones.

In the future, we plan to explore the option of performing the two steps of river and terrain generation in a multi-iteration cycle that gradually refines the landscape. This would resemble the real procedures of terrain generation more closely, and thus, may produce even more realistically looking and detailed results. Another option would be to increase the landscape's variety by adding more landscape features and region types such as oxbow lakes, wetlands, and cliffs. Lastly, parallelization of some calculations could improve the computational times further.

References

1. Balister, P., Balogh, J., Bertuzzo, E., Bollobás, B., Caldarelli, G., Maritan, A., Mastrandrea, R., Morris, R., Rinaldo, A.: River landscapes and optimal channel networks. Proceedings of the National Academy of Sciences **115**(26), 6548–6553 (2018). DOI 10.1073/pnas.1804484115

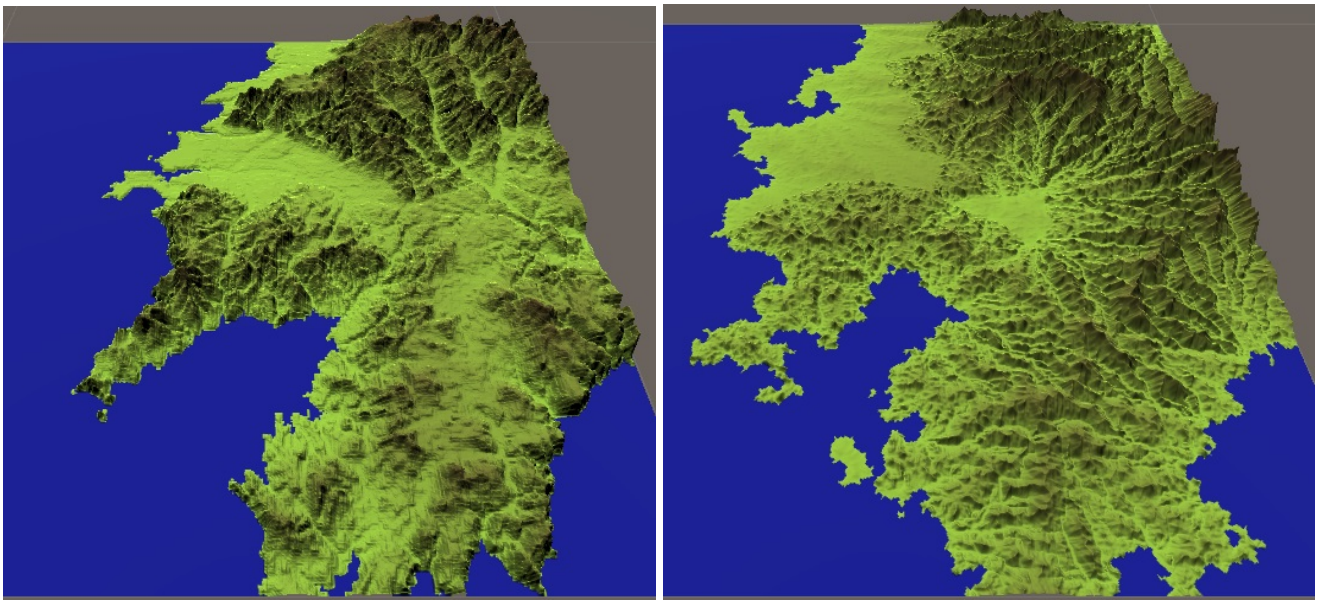

Fig. 14 Comparison of a real world's terrain (left) and our recreation (right), both visualized with meshes. Note that our recreation is quite similar.

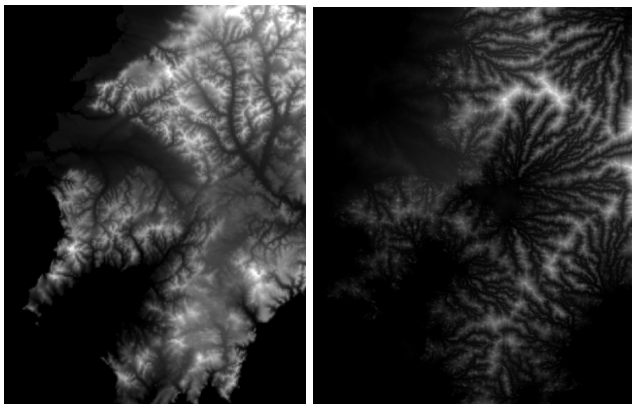

Fig. 15 Comparison of a real world's heightmap (left) and our recreated terrain's heightmap (right). Our system is able to recreate the fractal nature found in the real world.

2. Brown, R., Pasternack, G.: How to build a digital river. Earth-Science Reviews **194** (2019). DOI 10.1016/j.earscirev.2019.04.028
3. Carrara, F., Altermatt, F., Rodriguez-Iturbe, I., Rinaldo, A.: Dendritic connectivity controls biodiversity patterns in experimental metacommunities. Proceedings of the National Academy of Sciences of the United States of America **109**, 5761–6 (2012). DOI 10.1073/pnas.1119651109
4. Carraro, L., Bertuzzo, E., Fronhofer, E., Furrer, R., Gounand, I., Rinaldo, A., Altermatt, F.: Generation and application of river network analogues for use in ecology and evolution. Ecology and Evolution **10** (2020). DOI 10.1002/ece3.6479
5. Cordonnier, G., Braun, J., Cani, M.P., Benes, B., Galin, E., Peytavie, A., Guérin, E.: Large scale terrain generation from tectonic uplift and fluvial erosion. Computer Graphics Forum **35** (2016). DOI 10.1111/cgf.12820
6. Cortial, Y., Peytavie, A., Galin, E., Guérin, E.: Procedural tectonic planets. Computer Graphics Forum **38**, 1–11 (2019). DOI 10.1111/cgf.13614
7. Derzapf, E., Ganster, B., Guthe, M., Klein, R.: River networks for instant procedural planets. Computer Graphics Forum **30**, 2031 – 2040 (2011). DOI 10.1111/j.1467-8659.2011.02052.x
8. Fischer, R., Boeckers, J., Zachmann, G.: Procedural generation of landscapes with water bodies using artificial drainage basins. In: Computer Graphics International (2022). DOI 10.1145/3208159.3208184
9. Fischer, R., Dittmann, P., Weller, R., Zachmann, G.: Autobiomes: procedural generation of multi-biome landscapes. The Visual Computer pp. 1 – 10 (2020)
10. Gain, J., Marais, P., Straßer, W.: Terrain sketching. In: I3D '09, pp. 31–38 (2009). DOI 10.1145/1507149.1507155
11. Gain, J., Merry, B., Marais, P.: Parallel, realistic and controllable terrain synthesis. Computer Graphics Forum **34** (2015). DOI 10.1111/cgf.12545
12. Galin, E., Guérin, E., Peytavie, A., Cordonnier, G., Cani, M.P., Benes, B., Gain, J.: A review of digital terrain modeling. Computer Graphics Forum **38** (2019). DOI 10.1111/cgf.13657
13. Génevaux, J.D., Galin, E., Guérin, E., Peytavie, A., Benes, B.: Terrain generation using procedural models based on hydrology. ACM Transactions on Graphics **32**, 13 (2013). DOI 10.1145/2461912.2461996
14. Guérin, E., Digne, J., Galin, E., Peytavie, A., Wolf, C., Benes, B., Martinez, B.: Interactive example-based terrain authoring with conditional generative adversarial networks. ACM Transactions on Graphics **36** (2017). DOI 10.1145/3130800.3130804
15. Kelley, A., Malin, M., Nielson, G.: Terrain simulation using a model of stream erosion. In: ACM Siggraph Computer Graphics, vol. 22, pp. 263–268 (1988). DOI 10.1145/54852.378519
16. Lagae, A., Lefebvre, S., Cook, R., Derose, T., Drettakis, G., Ebert, D., Lewis, J., Perlin, K., Zwicker, M.: A survey

of procedural noise functions. Computer Graphics Forum **29** (2010). DOI 10.1111/j.1467-8659.2010.01827.x
17. Mei, X., Decaudin, P., Hu, B.G.: Fast hydraulic erosion simulation and visualization on gpu. pp. 47 – 56 (2007). DOI 10.1109/PG.2007.15
18. Michel, É., Emilien, A., Cani, M.P.: Generation of folded terrains from simple vector maps. In: Eurographics (2015)
19. Rigon, R., Rinaldo, A., Rodriguez-Iturbe, I., Bras, R., Ijjasz-Vasquez, E.: Optimal channel networks - a framework for the study of river basin morphology. Water Resources Research **29**, 1635–1646 (1993). DOI 10.1029/92WR02985
20. Rosgen, D.L.: A classification of natural rivers. Catena **22**, 169–199 (1994)
21. Stava, O., Benes, B., Brisbin, M., Krivanek, J.: Interactive terrain modeling using hydraulic erosion. In: Symposium on Computer Animation, pp. 201–210 (2008)
22. Talgorn, F.X., Belhadj, F.: Real-time sketch-based terrain generation. In: Computer Graphics International, pp. 13–18 (2018). DOI 10.1145/3208159.3208184
23. Teoh, S.: Riverland: An efficient procedural modeling system for creating realistic-looking terrains. In: Advances in Visual Computing, pp. 468–479 (2009). DOI 10.1007/978-3-642-10331-5_44
24. Zhang, H., Qu, D., Hou, Y., Gao, F., Huang, F.: Synthetic modeling method for large scale terrain based on hydrology. IEEE Access **4**, 6238–6249 (2016). DOI 10.1109/ACCESS.2016.2612700
25. Zhou, H., Sun, J., Turk, G., Rehg, J.: Terrain synthesis from digital elevation models. IEEE transactions on visualization and computer graphics **13**, 834–48 (2007). DOI 10.1109/TVCG.2007.1027