DOI: 10.15439/2023F2137
ISSN 2300-5963 ACSIS, Vol. 35

Simulating Large-Scale Topographic Terrain Features with Reservoirs and Flowing Water

Łukasz Błaszczyk, Michalina Mizura, Aleksander Płocharski, Joanna Porter-Sobieraj
0000-0002-9827-3562, 0009-0007-1683-6442, 0000-0002-7487-8153, 0000-0002-1411-475X
Warsaw University of Technology, Faculty of Mathematics and Information Science
ul. Koszykowa 75, 00-662 Warszawa, Poland
Email: {lukasz.blaszczyk, michalina.mizura.stud, aleksander.plocharski, joanna.porter}@pw.edu.pl

***Abstract*—The flow and accumulation of water are essential aspects when it comes to generating realistic terrains. In this article, we have set out to create a method for generating the distribution and levels of water in a virtual world. Our method fully simulates the water entering and exiting the system through rainfall, perspiration, and flowing out of the domain. Also, it simulates the phenomena of flow and accumulation in an iterative process. According to our observations, only allowing the user to influence the terrain and then simulating the placement of water bodies provides a natural result while preserving a high level of control.**

I. Introduction

THE rapid growth of the game and movie industries has generated a rise in demand for virtual worlds generation methods, especially when generating realistic natural environments. Modeling the environment by hand is, of course, still possible but only realistic when it comes to small parts of the vast virtual worlds that have become common. The workforce needed to create a whole extensive world by hand would be too substantial to consider these days. That is why the need for an automatic generation has drastically risen over the last decade to shorten this possibly tedious process.

The field in recent years has been moving in various directions depending on the use case being considered. Some algorithms focus purely on generating realistic natural environments from a humble set of parameters. While these methods usually provide very little control over the result, they are strongly based on geological knowledge (like tectonic plates) and climate data [1] and produce compelling results. These algorithms provide the foundation for most automatic world generation research. However, since the desired methods are supposed to replace modeling by hand, the methods at the forefront of interest usually provide much control over the result while still retaining a satisfying level of realism. The user may want to place mountain tops in certain specific places or keep a section of the environment flat. Unfortunately, the problem that arises seems to be that walking that thin line between control and realism is not so obvious and means that some natural processes might become neglected in the generation process.

We have observed that one of the environmental aspects that has a high chance of being treated like that is the placement and flow of water bodies in the result. While the methods usually aim for realistic placement of rivers and lakes, it is achieved mainly by some heuristic metrics rather than simulation loops. In most cases, the water bodies are just placed according to the positions specified by the user, and the environment is supposed to adjust independently to those requirements.

Since the flow and accumulation of water are essential aspects when it comes to keeping the veil of realism intact, we have set out to create a method for generating the distribution and levels of water in a virtual world. We have decided on a method that fully simulates the process of water entering and exiting the system (through rainfall, perspiration, and flowing out of the domain) and also simulates the phenomena of flow and accumulation in an iterative process. The method in itself does not offer much control over the result. However, the user can still use a terrain generation method which allows for controlling the shape of the terrain and place valleys and depressions in places where they want rivers and lakes to appear, and by applying our algorithm after that, the likelihood of meeting those water bodies placement expectations is very high. We have observed that only giving the user the ability to influence the terrain and then simulating the placement of water bodies naturally usually provides a very natural result while still preserving a high level of control based on our human intuition of where water is most likely to flow in a system of a given shape.

II. Related work

Terrain generation methods are usually grouped into two distinct categories based on their output. The first and more popular category is generating a heightfield representing the resulting terrain. In contrast, the second group of algorithms focuses on producing a three-dimensional output using various volumetric structures.

We first examine the former category, which, while having some apparent drawbacks – not being able to model things like caves and overhangs – usually provides the performance edge, which is invaluable for most iterative world creation pipelines [2]. One approach to the problem of 2D generation is constructing the terrain from primitive structures and defining their connections using a tree structure [3]. The primitives can be modeled by hand or generated from real-life data, for instance, using point cloud data from photogrammetry [4]. The method can yield compelling results, but the quality depends on the artist's skill in fitting those primitives together. While still cutting much modeling time, this approach cannot guarantee realistic results and will only provide partially automatic generation.

A significant subset of methods focuses on getting a more realistic result by employing geological knowledge. One of the aspects which can be used is the presence of tectonic plates since the process of forming mountains is influenced mainly by their movement and the placement of their borders. The position and shape of the plates can be generated from features that the user wants to appear in the final result, like the position of mountain tops and rivers [5]. The tectonic plate information can then be used to simulate the geological terrain folding process. Another example of using geological knowledge would be generating the resulting terrain using examples from the real world. Various distributions of geological parameters can be gathered for specific regions of the Earth, like the Himalayas or Norwegian fjords. Given some user input like positions and heights of mountain tops, the generation process can then try to recreate those distributions in the virtual environment [6]. This can produce convincing and realistic results, but since the output is majorly based on real-life data, it is limited to only resembling naturally occurring environments.

Rough sketches of the desired results can also guide the generation of terrains. The user could draw a 2D representation of the required scene using color-coded shapes or control points representing various topographical features like mountain ridges, plateaus, rivers, etc. [7], [8]. Unfortunately, this approach, like some of the methods mentioned above, suffers from very varied levels of realism in the resulting terrain based on the input data.

A good compromise between control and realism seems to be allowing the user to create maps of geological parameters that should be applied to the domain instead of specific topographical features that must appear in the resulting terrain. One such geological parameter is the tectonic uplift defining the growth rate of a particular point in the two-dimensional domain. The generation method can then simulate the growth of mountain ranges using the geological data provided while considering other factors like fluvial erosion [9].

Now we focus on the volumetric algorithms to briefly overview the rest of the generation methods spectrum. After adding dimension, a primitive-based modeling solution is still an excellent way to produce a convincing environment. It seems to be one of the most prominent methods for volumetric generation. One possible approach is basing the primitive structures on B-splines [10], which most artists are used to working with. The result becomes more flexible since it can model things like overhangs, allowing for smoother control but straying further from geological realism. The terrain could also be generated using a generalization of the previously mentioned primitive tree-based method [11]. Unfortunately, since the primitives must now be positioned across three dimensions, the work required to create desired scenes significantly increases.

There also exists a middle ground between the two representations – each point on a flat plane could hold a series of layers that are present above it together with their heights, representing a narrow column of the resulting terrain. This approach could either be used in conjunction with the standard two-dimensional methods or be extended with the ability to create a layer containing only air, allowing us to model things like caves and overhangs [12].

What most of the described methods have in common is that they treat the placement of water bodies in the result more like an afterthought rather than a significant step of the algorithm. Rivers and lakes are usually generated using heuristic methods to fit the generated environment, or their placement is dictated purely by the input data, which allows for setting their positions by the user almost entirely.

To design our algorithm, we needed to choose a terrain generation algorithm that would serve as a base for our water flow simulation. We have decided on using the method by Cordonnier at al. [9], which – apart from using the previously mentioned uplift maps – also considers the flow of water to be a significant factor in the process of erosion that is being simulated. This allowed us to use some of the water flow data structures already present in the solution as a jumping-off point for our method.

III. METHOD

A. *Starting off point*

The base method [9] operates over domain $\Omega \subset \mathbb{R}^2$ and aims to compute the function $h\colon \Omega \to \mathbb{R}$ representing the height of the terrain at each domain point. During the initialization step the method generates a terrain graph G defined over a discretization of the domain achieved by computing the Voronoi cells of points distributed over the domain using the Poisson distribution. In order to perform the fluvial erosion step the algorithm also computes a directed stream graph G_S, based on the same nodes as graph G, representing the water flow in the system, which serves as the foundation of our method.

Each node v in the graph G_S is represented by the cell's surface area a_v and a generating point p_v. The edges in the graph G_S represent the flow of water. Each node is connected to its lowest (in terms of height during this iteration) neighboring region, apart from points on the very edge of the domain where we assume the water flows out of the system. This creates disjointed components in the graph G_S (each in a form of a directed tree with the lowest - in terms of height - node forming the root), which in the original method are called lakes. Connection between these trees are stored in a separate lake graph G_L with edges defining the lowest connection between them - the minimum height at which one lake would start overflowing to the other.

The base method performs an additional step to merge the trees in G_S (representing lakes) into a complete graph. This is done by connecting them according to the connections in G_L. The algorithm allows itself to do that based on the assumption that all lakes are fully filled at all times – an assumption that we reject in the design of our method. That is why our solution is based on a disconnected stream graph; any mentions of it from now on will refer to this version.

At the base level of the solution, we have also incorporated different types of rock layers into the terrain representation [13]. Each point in the domain bears the characteristics of all layers below it with appropriate weights. This modification improves the overall realism of the result but does not influence the structure of the proposed water accumulation algorithm.

B. Algorithm modification

Algorithm 1 presents a general description of our modified fluvial erosion algorithm. A water accumulation step, performed during each iteration, and a lakes computation step have been added. Since a stream graph built for the current elevation of nodes is required for the correct computation of lakes, it is necessary to recreate this graph after the last change in elevation caused by erosion.

Algorithm 1 Base fluvial erosion algorithm [9] extended by adding water accumulation and lakes computation (underlined sections)

```
Require: $u$ {uplift map}, $i_{max}$ {maximum number of iterations} and $P$ {terrain sample points}
 1: compute terrain graph $G$
 2: for $i = 1$ to $i_{max}$ do
 3:     compute stream graph $G_S$
 4:     update accumulated water level in $G$
 5:     update $G_S$ with edges resulting from lake overflow
 6:     compute the new elevation values $h(v)$ after uplift and
        erosion for $v \in V(G)$
 7: end for
 8: compute stream graph $G_S$
 9: update accumulated water level in $G$
10: compute the lakes list $L$
11: determine the elevation values $h(p)$ for $p \in P$ by interpolating values $h(v)$ for $v \in V(G)$
12: return $h$ {elevation map} and $L$ {lakes list}
```

1) Water accumulation: Each vertex v of the terrain graph G stores information about the volume V_v of water collected on the part of the terrain represented by this vertex. The initial value of V_v is 0.

In this model, it was assumed that two factors had the most significant impact on the change in the accumulated water level: the increase in water resulting from precipitation and the evaporation phenomenon. Let us consider the first one. Due to the immense time scale of the simulation (millions of years), we assume that the amount of water that will accumulate at the bottom of the lake represented by its root v during a given iteration will be proportional to the time δt of the iteration and the basin area A_v, i.e.,

$$V_v^{rain}(t + \delta t) = \delta t \cdot rA_v(t + \delta t),$$

where r is a constant describing the amount of precipitation per unit area.

The evaporation phenomenon leads to a partial loss of water in each iteration. We assume that the amount of water that has evaporated will be proportional to the iteration time and the area of the considered lake. Studies on the characteristics of natural lakes [14] have shown that the relationship between the volume V of a lake and its water surface S is approximately $V \sim S^{6/5}$. Thus, the volume of water lost can be written as:

$$V_v^{evap}(t + \delta t) = \delta t \cdot eS_v = \delta t \cdot eV_v^{5/6}(t),$$

where e is a constant controlling the evaporation rate. Using this approximation allows us to estimate the water loss due to evaporation without repeatedly determining the actual areas of the lakes.

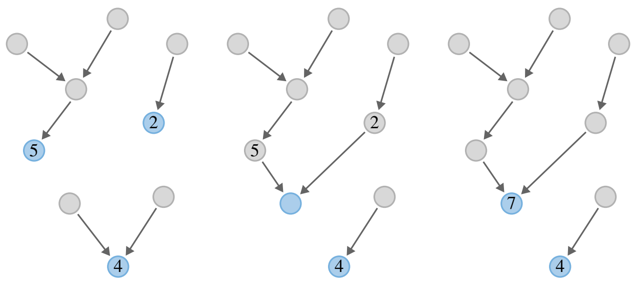
The formula for changing the volume of water during an iteration of δt is then:

$$\Delta V_v(t + \delta t) = \delta t \left(rA_v(t + \delta t) - eV_v^{5/6}(t) \right).$$

It is also worth noting that because each iteration of the stream graph G_S contains different connections, it is necessary to traverse all the vertices belonging to a given subtree and move the water accumulated in them so far to the current root. This step is performed before calculating the change in water volume so that water accumulated at other vertices can also participate in evaporation. The water-shifting process is illustrated in Figure 1.

In the stream graph G_S created during each iteration of the algorithm, roots of trees contained in G_S do not have receivers – water does not flow from them to another node. If the tree's root is at the edge of the terrain, we can assume that rainfall water flows out of the domain. Otherwise, the water should stagnate in the area of the tree (starting from the root), creating a lake.

2) Computing and merging lakes: The key element of the described modification is the algorithm for determining the water level in each lake. The non-trivial nature of this task results from the fact that more water can be accumulated in the lake than the surrounding area can hold. Therefore, a solution is needed that will allow for modeling the flow of water between lakes, as well as combining several lakes into



(a) i-th iteration (b) $(i + 1)$-th iteration (c) result

Fig. 1. The process of water shifting to its current roots in a stream graph. The numbers in the vertices indicate the volume of water accumulated in them. Different edge structure between iterations is caused by the erosion process which changes the height of the nodes resulting in the need to update the stream graph.

Algorithm 2 Determining the water level in lakes

```
Require: $G_S$ {stream graph}
 1: $G_L \leftarrow$ lakes graph determined by $G_S$
 2: $L \leftarrow V(G_L)$ {merged lakes set}
 3: for all lakes $k$ in $V(G_L)$ do
 4:   $v \leftarrow$ root of $k$
 5:   $V_k^{rem} \leftarrow V_v$ {the water remaining to fill up is all the water collected in the root}
 6:   $V_k^{acc} \leftarrow 0$ {water accumulated in a given lake}
 7:   $l_k \leftarrow$ neighbor of $k$, to which the lowest passage from $k$ leads
 8:   $V_k^{miss} \leftarrow$ the amount of water needed to reach the crossing height $(k, l_k)$
 9: end for
10: while the flow is non-zero or no lakes have merged do
11:   $s \leftarrow$ network source, $t \leftarrow$ network sink
12:   $N \leftarrow$ empty lake network; $V(N) = \{L, s, t\}$
13:   for all lakes $k$ in $L$ do
14:     if $V_k^{rem} > 0$ then
15:       append to $N$ arc $(s, k)$ with capacity $V_k^{rem}$
16:     end if
17:     if $V_k^{miss} > 0$ then
18:       append to $N$ arc $(k, t)$ with capacity $V_k^{miss}$
19:     else
20:       append to $N$ arc $(k, l_k)$ with capacity $\infty$
21:     end if
22:   end for
23:   $SCC \leftarrow$ set of strongly connected components in $N$
24:   for all strongly connected components $H$ in $SCC$ such that $|H| > 1$ do
25:     collapse the vertices belonging to $H$ into one pool $m$
26:     determine $l_m$ and $V_m^{miss}$
27:     remove the component lakes and add a new merged lake $m$ to $L$
28:   end for
29:   $F \leftarrow$ maximal flow in $N$
30:   for all arcs coming from the source $(s, k)$ in $F$ do
31:     subtract the flow value at $(s, k)$ from $V_k^{rem}$
32:   end for
33:   for all arcs entering the sink $(k, t)$ in $F$ do
34:     add the flow value at $(k, t)$ to $V_k^{acc}$
35:     subtract the flow value at $(k, t)$ from $V_k^{miss}$
36:   end for
37: end while
38: for all lakes $k$ in $V(G_L)$ do
39:   $h^{acc} \leftarrow$ water level based on $V_k^{acc}$
40: end for
41: return $G_L$ {graph of lakes supplemented with information about the water level $h^{acc}$}
```

one. The pseudocode of the developed algorithm is described in Algorithm 2.

a) Computation of water volume and surface height: During the algorithm, it is necessary to determine the volume of water required to reach a certain surface height. We can

Algorithm 3 Determining the volume of water needed to reach the given height of the surface

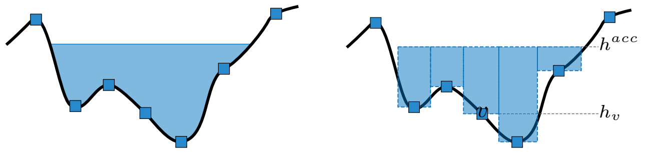
```
Require: $G_S$ {stream graph}, $k$ {lake}, $V_k^{acc}$ {volume of water accumulated in a given lake} and $h^{target}$ {target surface height}
  $V^{total} \leftarrow 0$
  $Q \leftarrow$ empty queue
  for all lakes $l$ forming part of the combined lake $k$ do
    if root of $l$ is below $h^{target}$ then
      append root of $l$ to $Q$
    end if
  end for
  while $Q$ is not empty do
    remove vertex $v$ from $Q$
    $V^{total} = V^{total} + a_v \cdot (h^{target} - h_v)$
    for all nodes $w$ in the set of children of $v$ do
      if $h_w < h^{target}$ then
        append $w$ to $Q$
      end if
    end for
  end while
  $V_k^{miss} \leftarrow V^{total} - V_k^{acc}$
  return $V_k^{miss}$ {the amount of water needed to reach a given height}
```

approximate the volume of the lake by the sum of prisms whose bases are the Voronoi cells of individual vertices of the terrain graph located underwater, and the height is the difference between the height of the surface h^{acc} and the height of the vertex h_v. The volume approximation method is illustrated in Figure 2 and described in Algorithm 3.

It is worth noting that the lake for which calculations are made can be either a single tree in the stream graph or a combination of several trees into one merged lake.

At the end of the lake computation algorithm, a reverse operation is required, i.e., calculating the height of the water surface based on the volume of collected water. The method chosen for this purpose consists of gradually flooding the tops of the lake to increase the height until the lowest unflooded top is above the existing water surface. A detailed description of this step is provided in Algorithm 4.

b) Finding strongly connected components: Strongly connected components are essential in constructing a network describing the water flow between lakes. They represent



(a) The actual volume of water in the lake (b) Approximating the volume of water by the sum of the prisms

Fig. 2. Cross-section of an example lake filled with water to the level of h^{acc}

Algorithm 4 Determining the height of the lake surface for a given volume of water

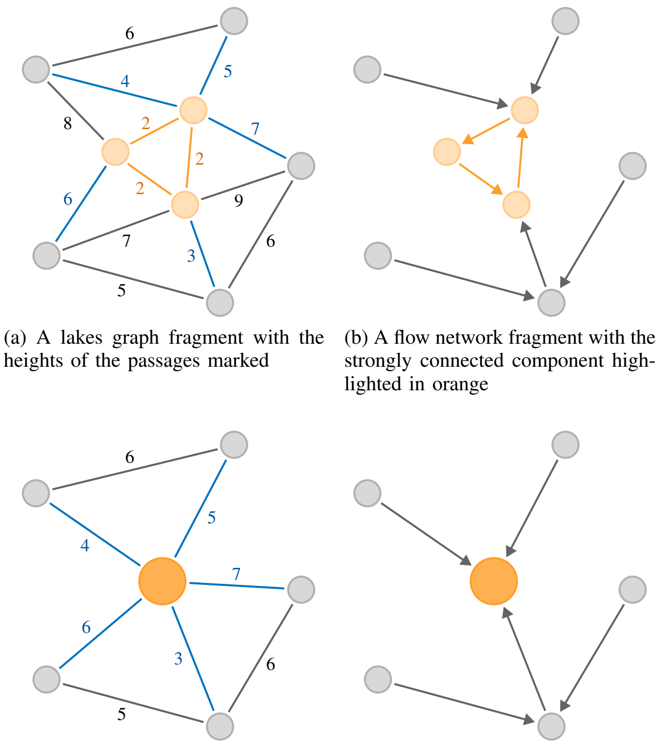
```
Ensure: $G_S$ {stream graph}, $k$ {lake} and $V_k^{acc}$ {volume of water accumulated in a given lake}
  $Q \leftarrow$ empty priority queue {priority – lowest height}
  $V_k^{below} \leftarrow 0$ {volume of land under the lake}
  $A_k^{acc} \leftarrow 0$ {surface area}
  $h_k^{acc} \leftarrow \infty$
  for all lakes $l$ forming part of the combined lake $k$ do
    append root of $l$ to $Q$
  end for
  while $Q$ is not empty and the lowest vertex of $Q$ is below $h_k^{acc}$ do
    remove the lowest vertex $v$ from $Q$
    $A_k^{acc} \leftarrow A_k^{acc} + a_v$
    $V_k^{below} \leftarrow V_k^{below} + a_v \cdot h_v$
    $h_k^{acc} \leftarrow \frac{V_k^{acc}+V_k^{below}}{A_k^{acc}}$
    for all nodes $w$ in the set of children of $v$ do
      if $h_w < h_k^{acc}$ then
        append $w$ to $Q$
      end if
    end for
  end while
  return $h_k^{acc}$ {surface height}
```

groups of two or more lakes connected by passages at the same height and filled with water at least up to that height. Adding water to the system of lakes thus connected would result in a combined rise in the level of all the constituent lakes. To model this behavior, lakes belonging to the strongly connected component are represented by one vertex when determining the flow. This means that the water flow network and the lake graph G_L must be modified at each iteration, as shown in Figure 3.

Tarjan's algorithm [15] was used to find strongly connected components in the directed lake graph G_L. It allows us to determine all such components in time $O(V+E)$. Since each vertex except sources and sinks has precisely one outgoing arc in the considered network, the complexity, in this case, will be $O(M)$, where M is the number of lakes.

c) Computing the maximum flow: Flow in a network of lakes allows the determination of the amount of water flowing out of each lake and into each lake. The arcs from the source to the lakes with the capacity of the collected water model the water that directly flows into the basin of a given lake due to precipitation. The arcs from the lakes to the estuary with a capacity representing the volume to the lowest passage represent the water accumulating in the basin. The arches between the lakes allow water to flow through passages.

Push-relabel is the chosen algorithm for determining the maximum flow in the network. An appropriate implementation of this algorithm [16] allows obtaining a complexity of $O(V^2\sqrt{E})$, which in the case of the considered network is $O(M^2\sqrt{M})$.



(a) A lakes graph fragment with the heights of the passages marked

(b) A flow network fragment with the strongly connected component highlighted in orange

(c) A lakes graph fragment after the collapse of the strongly connected component. Only those edges incident with the component, which have the smallest height, are left

(d) A flow network fragment after the collapse of the strongly connected component

Fig. 3. Modification of the lakes graph and the water flow network as a result of replacing the strongly connected component with one merged lake

d) Optimizations: The first optional optimization of the lake computation algorithm is the step of joining pairs of lakes for which the lowest passage is at the same level as both of their roots. This situation occurs primarily in the first iterations of the erosion algorithm when many vertices are at the same level. This optimization will connect many lakes that consist of only one vertex, significantly reducing the lake network's size.

The second optimization used consists of the lakes' initial filling with the water collected in them. This step is done after executing the 8 line in Algorithm 2. For lake k, V_k^{acc} becomes $\min(V_k^{rem}, V_k^{miss})$, and V_k^{rem} and V_k^{miss} will be reduced by same amount. This means that if more water has accumulated in the lake than is necessary to fill it, then already in the first iteration of the lake computation algorithm, water from it can flow to the remaining reservoirs.

3) Computational complexity: The water accumulation step involves traversing the stream graph to move water from the non-root vertices to the roots and determining a new value for each lake's water volume. Hence the complexity of this step is $O(N+M) = O(N)$, where N is the node count in stream graph G_S. Since each iteration of the basic fluvial erosion algorithm requires $O(L \cdot N + M \log M)$ operations, where L

is the terrain layer count, adding a water accumulation step does not affect the complexity of the iterations.

In order to determine the complexity of the lake determination algorithm, all steps described in the Algorithm 2 should be considered. Determining the lakes graph in line 1 has a complexity of $O(N)$ due to the need to traverse all edges of the terrain graph and its planarity. The loop in line 3 executes M times, calling Algorithm 3 for each lake. However, since within Algorithm 3, each vertex of the terrain graph will be queued at most once, the total execution time for all iterations of this loop will be $O(N)$. The situation for the loop in line 38 is analogous, so this fragment also has a complexity of $O(N)$.

Let us now consider the execution time of one iteration of the lake determination algorithm. The complexity of the loops in 13, 30 and 33 is $O(M)$. Finding strongly connected components using Tarjan's algorithm also requires $O(M)$ operations. Collapsing them into one vertex is also $O(M)$ complex. This step involves traversing all the vertices of the found components and updating the arcs in the network incident with those vertices. We assume that we can find a specific arc and then modify or delete it in constant time, which is possible, for example, by using hash tables as collections of arcs. The most demanding operation, with a complexity of $O(M^2\sqrt{M})$, is finding the maximum flow in the network.

The last key issue is the number of iterations performed by the lake determination algorithm. The chosen stopping condition means that in each iteration except the last one, there must be either a combination of at least one strongly connected component with a cardinality greater than 1 into one lake or a non-zero flow.

Each iteration in which a newly merged lake is created reduces the number of network vertices representing the lakes by a minimum of one. This means that at most $M-1$ of such iterations is possible.

The lake merging operation is the algorithm's only element that modifies the network structure between successive iterations. This means that if the lakes do not connect, the network in the next iteration will be the same network with the volumes of the arcs minus the value of the last maximum flow, resulting in zero flow. There can therefore be at most M iterations in which the lakes do not merge. Hence the total number of iterations is limited by $O(M)$.

The total computational complexity of the lake determination step is $O(N+M^3\sqrt{M})$. It is worth noting, however, that the number of vertices in the network decreases with each strongly connected component collapse, so in practice, later iterations of the water level determination algorithm run faster.

4) Computation of lakes and the basic erosion algorithm: In our modified fluvial erosion algorithm, water overflow between lakes is a separate step independent of lake overflow during the iteration. This approach proved necessary to preserve the realistic and varied resulting terrain.

The first attempts to extend the erosion algorithm consisted of replacing the original overflow of lakes with a model that would consider water stagnating in depressions. However, this approach did not produce the expected results. Since the terrain was almost flat in the first iterations, the water stagnated at numerous points and led to the formation of craters. It was, therefore, necessary to separate the overflowing and the computation of lakes. The first of these stages allows us to shape the relief, while the second only determines which places in the existing area should be flooded with water.

5) Elevation computation for vertices on the boundary of the domain: The original version of the fluvial erosion algorithm described in [9] assumes that vertices of the terrain graph located on the edge of the domain never change the height. As a result, the edge of the resulting terrain is always at 0. As the rest of the land is uplifted, this leads to the formation of a steep slope along the entire length of the border.

This problem can be eliminated by determining the vertices' elevations on the domain's border in the same way as for the other vertices. However, since some of these vertices do not have receivers, it is necessary to modify the original formula [9, eq. (2)].

The proposed modification consists in replacing the value $h_w(t+\delta t)$ with 0 and replacing the distance $\|\mathbf{p}_v - \mathbf{p}_w\|$ with a large constant, for example, the domain size. This is equivalent to introducing artificial vertices of constant elevation 0 into the terrain graph, which are ignored when determining rivers and lakes. So the formula for vertices without a receiver is:

$$h_v(t+\delta t) = \frac{h_v(t) + \delta t u_v}{1 + \dfrac{k\sqrt{A_v}}{D}\delta t},$$

where D is the domain size.

Thanks to this modification, the resulting terrain has a natural-looking edge. It gives the impression of a fragment cut out from a larger area, allowing it to be used in various applications without additional processing. In addition, the terrain more accurately reflects the features marked on the uplift speed map, giving the user more control over the outcome.

IV. Results analysis

The new proposed version of the whole terrain generation algorithm has been implemented in C++ as an *Unreal Engine 5* plugin. The plugin allows the user to generate terrain based on required input parameters and save it in one of the available formats. All reported results were obtained on a PC with an AMD Ryzen 7 1700 3.0 GHz and 16 GB RAM, supplied with an NVIDIA GeForce GTX 1080.

The algorithm's parameters allow the user to control the amount of water accumulating in lakes as well as decide the threshold for considering a stream of water to be a river. Figure 4 shows an example of the result of our method with generated lakes and rivers on display.

Figure 5 demonstrates the influence of the water accumulation parameters on resulting lakes. An increase in precipitation creates an increase in water flow in the system, which raises the water levels of the lakes, possibly even causing overflow

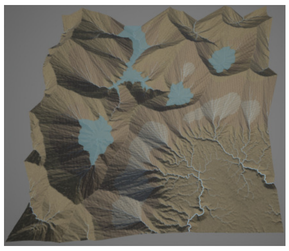
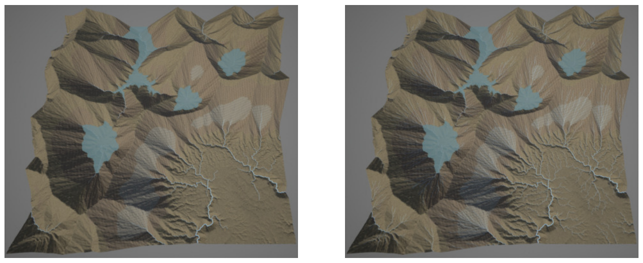
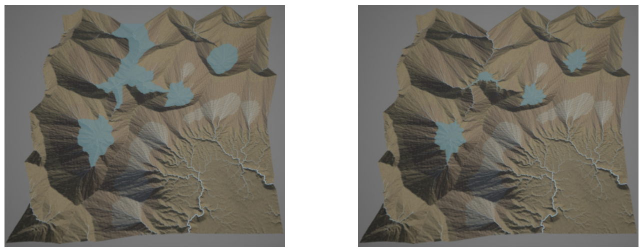


Fig. 4. Terrain with generated lakes and rivers



(a) Terrain with a high river threshold (b) Terrain with a low river threshold

Fig. 6. The impact of the river threshold parameter on the resulting terrain



(a) Terrain with increased precipitation (b) Terrain with increased evaporation rate

Fig. 5. The impact of water accumulation parameters on the resulting terrain

and joining some of them together. On the other hand, increasing the evaporation rate causes the water levels to fall, resulting in some of the lakes completely drying out.

Figure 6 shows the river network for different inclusion threshold levels. Setting a high value for the threshold results in highlighting only the most pronounced rivers in the basin, while keeping the value low allows us to keep even the small streams in the scene.

The computation time of the algorithm has been tested for multiple input terrain parameter presets:

- *Basic* – a constant terrain uplift on the whole domain
- *Perlin* – uplift of the domain defined by 2D Perlin noise
- *MountainSimple*, *MountainSteep* – uplift map typical for mountain regions; the second preset allows for steeper slopes
- *MountainLayers* – uplift map typical for mountain regions but with each layer subdivided into 5 additional sublayers

The key aspect of the algorithm is the time required for generating the terrain (Figure 7). The performed tests show that generating the terrain in low resolution (a 5 km $\times$ 5 km region with mesh vertices about 2.5 m from each other) takes from a dozen to several dozen seconds (Figure 7a). Such a resolution is enough to give the user a general idea of the topographical aspects of the resulting terrain. It allows for a quick iterative process of refining the result. The highest resolution (a 5 km $\times$ 5 km region with mesh vertices about 0.5 m from each other) in most cases computes in less than 30 minutes (Figure 7c) and allows the user to glimpse the precise shape of the result.

In one specific case, the time is substantially lengthened. When the uplift map is constant, the terrain remains mostly flat, resulting in a significant amount of puddles instead of a handful of big lakes resulting in the apparent rise of processing time (Fig. 7c) caused by the parts of the method that are dependant on the number of lakes. The other parameter set that was an outlier in terms of processing time, but this time by not that big of a margin, was the Mountain Layers preset. This is caused by the extended calculation time of the erosion step since multiple layers need to be considered.

It is worth noting that since the node and edge counts in the terrain graph are not dependent on the water accumulation input data but only on the size of the terrain and the sampling rate, the execution time of the lake generation process is invariant to different input parameter sets.

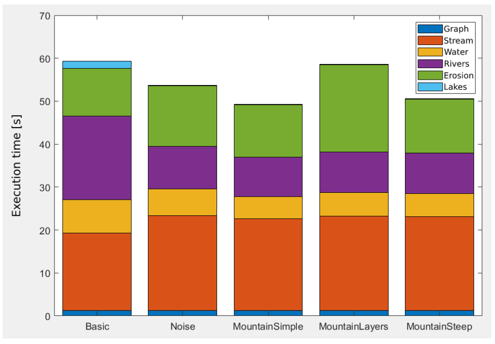
The extension of the terrain generation algorithm by adding the water accumulation and lake generation step impacts the total processing time marginally. The only exception is the *Basic* preset. However, since it is an artificial example created only for the purposes of testing the processing time and the method aimed at generating realistic environments, this result can be disregarded. For the highest resolution examples presented in the paper, the additional lake calculations only take up from 15% to 23% of the total computation time.

V. Conclusion

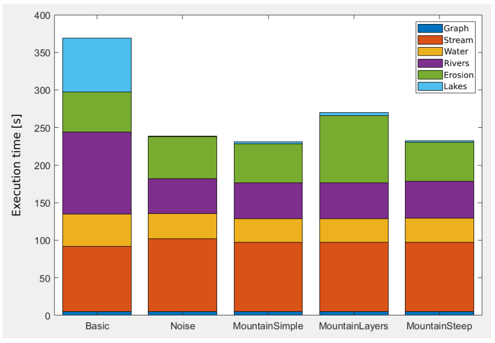
The modification presented in this paper to the base fluvial erosion algorithm [9] can allow the user to obtain diversified terrains in a few seconds to several dozen minutes. While the base method was only capable of generating terrains consisting of topographical features such as mountains, hills, plains the extension developed by us makes it possible to also add bodies of water to that list – rivers and lakes. This creates a much more natural looking environment since not only do we end up with more diverse features in the result but all of these features are also simulated simultaneously, interacting with each other during the generation process and in turn leading to a more lifelike result.

Rivers and lakes are created fully automatically. Rivers flow from higher to lower terrain points and naturally join together. Lakes form in natural depressions where water accumulates. The user can suggest the desired locations of water reservoirs by using the uplift velocity map.

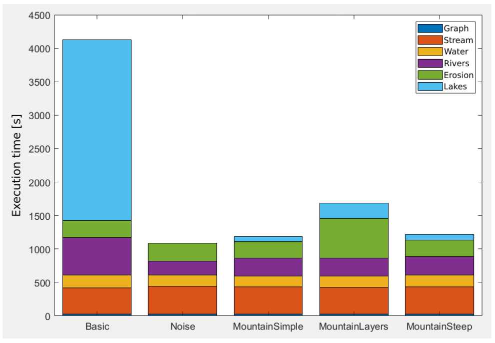
During further research, the presented new version of the algorithm, could also allow us to diversify the terrain's shape by changing the erosion result for submerged vertices. Such



(a) low resolution (5km×5km, vertices distance≈2.5m) - 300 iterations



(b) medium resolution (5km×5km, vertices distance≈1.5m) - 300 iterations



(c) high resolution (5km×5km, vertices distance≈0.5m) - 300 iterations

Fig. 7. Execution time results for different resolutions

a change could depend on a total change in height for all nodes belonging to a given lake or the phenomenon of sedimentation. It is worth noting that in the described version of the modified algorithm, it is not necessary to compute lakes after each iteration. However, any method improvement that would affect the erosion result for underwater vertices will also require information about the water level during the algorithm's evaluation.

Further improvements of the method may also relate to the user's level of control over the final terrain – for instance the ability to determine the location of water reservoirs. Currently, the user can suggest where rivers and lakes should appear using the uplift map, but their occurrence is not guaranteed. An alternative is to add the option of forcing some connections in the flow graph, freezing the height of selected vertices, or imposing additional conditions on them, which would facilitate the insertion of selected topographic features. This however, could lead to some loss of realism over more control given to the user.

Acknowledgment

The research was supported by Warsaw University of Technology faculty grant.

References

[1] J.-D. Champagnac, P. Molnar, C. Sue, and F. Herman, "Tectonics, climate, and mountain topography," *Journal of Geophysical Research: Solid Earth*, vol. 117, no. B2, 2012. doi: 10.1029/2011JB008348

[2] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes, "A survey on procedural modelling for virtual worlds," *Computer Graphics Forum*, vol. 33, no. 6, pp. 31–50, 2014. doi: 10.1111/cgf.12276

[3] J.-D. Génevaux, E. Galin, A. Peytavie, E. Guérin, C. Briquet, F. Grosbellet, and B. Benes, "Terrain modelling from feature primitives," *Computer Graphics Forum*, vol. 34, no. 6, pp. 198–210, 2015. doi: 10.1111/cgf.12530

[4] M. Luckner and K. Rzążewska, "3D model reconstruction and evaluation using a collection of points extracted from the series of photographs," in *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, 2014. doi: 10.15439/2014F304 pp. 669–677.

[5] E. Michel, A. Emilien, and M.-P. Cani, "Generation of folded terrains from simple vector maps," in *Eurographics 2015 short paper proceedings*. The Eurographics Association, 2015. doi: 10.2312/egsh.20151019

[6] O. Argudo, E. Galin, A. Peytavie, A. Paris, J. Gain, and E. Guérin, "Orometry-based terrain analysis and synthesis," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–12, 2019. doi: 10.1145/3355089.3356535

[7] D. B. Adams, "Feature-based interactive terrain sketching," Master's thesis, Brigham Young University, 2009. [Online]. Available: hdl.lib.byu.edu/1877/etd3221

[8] S. T. Teoh, "River and coastal action in automatic terrain generation," in *Proceedings of the 2008 International Conference on Computer Graphics and Virtual Reality*, 2008, pp. 3–9. [Online]. Available: citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=316be57e56662a0113a5678eb29dd5b3b951694a

[9] G. Cordonnier, J. Braun, M.-P. Cani, B. Benes, E. Galin, A. Peytavie, and E. Guérin, "Large scale terrain generation from tectonic uplift and fluvial erosion," *Computer Graphics Forum*, vol. 35, no. 2, pp. 165–175, 2016. doi: 10.1111/cgf.12820

[10] M. Becher, M. Krone, G. Reina, and T. Ertl, "Feature-based volumetric terrain generation and decoration," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 2, pp. 1283–1296, 2019. doi: 10.1109/TVCG.2017.2762304

[11] A. Paris, E. Galin, A. Peytavie, E. Guérin, and J. Gain, "Terrain amplification with implicit 3d features," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 5, pp. 1–15, 2019. doi: 10.1145/3342765

[12] A. Peytavie, E. Galin, J. Grosjean, and S. Mérillou, "Arches: a framework for modeling complex terrains," *Computer Graphics Forum*, vol. 28, no. 2, pp. 457–467, 2009. doi: 10.1111/j.1467-8659.2009.01385.x

[13] G. Cordonnier, M.-P. Cani, B. Benes, J. Braun, and E. Galin, "Sculpting mountains: Interactive terrain modeling based on subsurface geology," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 5, pp. 1756–1769, 2017. doi: 10.1109/TVCG.2017.2689022

[14] B. B. Cael, A. J. Heathcote, and D. A. Seekell, "The volume and mean depth of Earth's lakes," *Geophysical Research Letters*, vol. 44, no. 1, pp. 209–218, 2017. doi: 10.1002/2016GL071378

[15] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972. doi: 10.1109/SWAT.1971.10

[16] J. Cheriyan and S. N. Maheshwari, "Analysis of preflow push algorithms for maximum network flow," *SIAM Journal on Computing*, vol. 18, no. 6, pp. 1057–1086, 1989. doi: 10.1137/0218072