Procedural Content Generator for Generating a 3D Graphical Representation of the River Model

Alžbeta Sklenáriková   
*Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,*  
*Technical University of Košice*Letná 1/9, 042 00 Košice, Slovakia  
alzbeta.sklenarikova@fei.tuke.sk

Heidar Khorshidiyeh  
*Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,*  
*Technical University of Košice*Letná 1/9, 042 00 Košice, Slovakia  
heidar.khorshidiyeh@tuke.skBranislav Madoš  
*Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,*  
*Technical University of Košice*Letná 1/9, 042 00 Košice, Slovakia  
branislav.mados@tuke.sk

Peter Poprík  
*Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,*  
*Technical University of Košice*Letná 1/9, 042 00 Košice, Slovakia  
peter.poprik@tuke.skNorbert Ádám  
*Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,*  
*Technical University of Košice*Letná 1/9, 042 00 Košice, Slovakia norbert.adam@tuke.sk

Lukáš Tomaščík  
*Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,*  
*Technical University of Košice*Letná 1/9, 042 00 Košice, Slovakia  
lukas.tomascik@tuke.sk

*Abstract*—This work deals with procedural content generation, specializing in content generation in the fields of computer graphics and computer games. Even more precisely, it deals with the procedural generation of three-dimensional polygonal surface models of rivers. In the introductory part, the paper provides an overview of related procedural content generation principles using noise generating functions and more complex simulation techniques. Then, it presents a conceptual proposal for the procedural generation of a river model from the point of view of modeling its bed using the designed concept of control tiles. The generator's goal is to provide a tool for creating river models that can be integrated into the environment of 3D computer games.

Keywords—computer graphics, computer games, procedural content generator, polygonal surface model, model of the river

# Introduction

Procedural content generation (PCG) can be defined as generation of data based on specified rules, when algorithmic creation of content combines human-generated content with computer-generated randomness using defined rules [1].

Implementation of PCG is the standard in the field of computer graphics, especially in the industry of computer games, where it can be used to create assets, whole open worlds, height and collision maps, and also more complex content like spawn behavior. Looking at the range of possibilities related to creation of graphical content, there is no doubt that the ambitious vision of „a technology that unites artists and scientists” [2], shared by the engineers behind the creation of computer graphics in the 1960s, has come true.

In computer games, using PCG extends the playability time of the given product. It can also increase the ability to customize game content based on various aspects, such as the skill level of the character the player is playing [3]. Some games may not rely on PCG algorithms directly, in play-time, but offer their players the so-called „infinity module” [4], which generates, e.g., an infinitely large country.

The main advantages of using reliable PCG algorithms include saving time by generating content compared to manually inserting it into the game world, as well as the possibility of expanding the use of the generator to create various elements, not only those for which it was originally intended. There is also the certainty that once the generation rules are established, the PCG algorithm can follow them better than if human produced the given content.

One of the main disadvantages of procedural generation in games is a risk of losing control over how the game world looks like and what is happening in this world. Testers in the development phase cannot check every single iteration of procedurally generated content, so it is easy for bugs to remain hidden until the game is released and enough players have tried the game [5]. The development time of the PCG algorithms that generate the majority of game content cannot be underestimated. Any change in the project's direction may result in the necessity of comprehensive reforms of the already written code [6].

One of the visually very attractive content in computer games or virtual reality is 3D representation of rivers and water in general. The procedural generation of river models comprises a relatively large area of knowledge that combines several disciplines.

The reasons for creating models of rivers, or their parts, from a broader point of view, range from the preparation of environmental components for games, virtual reality, or movie scenes in the entertainment industry to scientific simulations, analysis, and predictions. Procedural generation algorithms may not always be part of a published game. They can be used to help in the design phase of game content. For example, when creating a city in Skyrim, a large number of objects are procedurally generated, which designers further modify into their final form [7].

River can be simulated in terms of the movement of liquids or the formation of the bank's surface and/or bottom. Models can respect the real physics of processes that are influencing rivers and therefore consume enormous amounts of computing resources or focus on conveying an aesthetic experience, ideally in real time and with the smallest time and memory complexities.

Algorithms and procedures for creating rivers in computer graphics and simulation differ concerning the expected goal of their generation. Thus, it is important to determine whether the goal is only to make the river appear credible or whether the logic of the calculations of its output should reflect real physical laws. Procedural algorithms allow the implementation of a wide range of outputs based on only a small variation of the input parameters. However, physically oriented algorithms are the only ones that provide results reflecting the mathematical regularity of the simulated phenomenon in reality. Their weakness is the lack of control over the generated output [8].

# Related Works

A river is a natural one-way stream of water flowing downstream under the influence of gravity from its source into the sea, lake, or other river [9]. The size of the river flow mainly depends on the depth of the river, i.e., the distance of the water level from the bottom of the basin, and its flow rate, i.e., the volume of water that flowed through the transverse profile of the stream bed [10]. In the procedural creation of natural surfaces, the question arises about how to express the rules of various natural patterns using algorithm.

One of the most effective tools for achieving this is procedural noise functions. Noise introduces random fluctuations within a given signal, where variations in intensity and distribution can create multi-dimensional patterns that emulate the complexity of natural textures. Procedural noise has many advantages: it is typically very fast to evaluate, often allowing evaluation of complex and intricate patterns on-the-fly, and it has a very low memory footprint, making it an ideal candidate for compactly generating complex visual details [11].

A blurry image of a cell

Description automatically generated

#### a) b)

**Fig. 1** Visualization of a) Perlin noise and b) Worley noise.

One of the first algorithms for generating realistic textures was Perlin noise [12]. This type of gradient noise interpolates smoothly between values, producing a continuous, random-looking effect. It is popular for creating textures that resemble natural patterns.

Perlin noise (Fig. 1a) is commonly used in the procedural generation of organic game content. The algorithm is used for many types of uneven materials and textures, which can be used to generate terrain or effects resembling the movements of fire, water, or clouds [13].

Worley noise (Fig. 1b), also known as Cellular noise, is an algorithm used to generate diverse patterns. Compared to Perlin noise, Worley noise is computationally more expensive, as it requires calculating distances between all points. However, it has the advantage of producing interesting patterns after a single iteration, whereas other noise functions often require multiple iterations to achieve similarly complex results [14].

Worley introduced this algorithm as a new basis function which complements Perlin noise, relying on the partitioning of space into a random array of cells [14].

Worley noise generation begins by dividing 2D, 3D, or higher-dimensional space into a grid, assigning each cell a randomly placed "seed" point. This setup is particularly effective with isotropic distributions. One example is the Poisson distribution, which divides space into a grid of identical tiles, with one point per tile [14], as can be seen in Fig. 2.

The algorithm then calculates the distance to the n-nearest neighboring points for each location, creating visually intriguing patterns.

A grid of red dots

Description automatically generated

**Fig. 2** Poisson distribution grid with “seed” points.

For a moving continuum (it means a fluid or a deformable solid substance), there are two approaches to describe their motion: The Lagrangian point of view and the Euler point of view [15].

The Lagrangian viewpoint, named after the French mathematician Lagrange, considers the continuum a particle system. Each point in the fluid or the solid is labeled as a separate particle with position x and velocity u. Discrete particles can be thought of as molecules of a given structure, and this simulation method is mainly used for solid substances with a discrete number of particles [16][17].

Euler's point of view, named after the Swiss mathematician Euler, uses a point of view where instead of observing every particle in the overall structure, the focus is on the behavior of selected factors (e.g., density or speed). The Navier-Stokes equations describe how a moving fluid's velocity, pressure, temperature, and density are related. These equations consist of a time-dependent continuum equation for the conservation of mass, three time-dependent equations for the conservation of momentum, and a time-dependent equation for the conservation of energy [20][21].

Despite their wide practical use, the theoretical elucidation of these equations is incomplete - and therefore, the proof of their existence is among the "seven problems of the millennium," declared by the Clay Mathematical Institute [22].

The MAC grid (Marker And Cell grid) was first published in 1965 by the American theoretical physicists Francis H. Harlow and J. Eddie Welch [23]. This method has become one of the most popular ways to simulate fluid flow in computer graphics. It is a "graded" grid that combines several types of nodal points at different geometric positions. This type of grid, complex at first glance, enables a clear and precise formulation of results for partial differential equations with finite differences, including the Navier-Stokes equations [24].

The level set method for simulating a moving surface was formulated in 1987 by S. Osher and J.A. Sethian [25]. To this day, a wide spectrum of its use is recorded in computer graphics – from special movie effects and visualizations, to computer vision or capture of multiphase dynamic flows of liquids [26].

The level set method allows greater control over the surface of the liquid. Thus, the simulation can get closer to the real behavior of the river level - with minimal movement and lower speed, the river level looks smooth, and on the contrary, with a faster river flow, the level will behave turbulently to the point of "torn" [27].

Interpolation and extrapolation are two commonly used types of mathematical predictions to estimate hypothetical values. While interpolation predicts values ​​that are already somewhere within the existing set of function values, extrapolation is used to predict values ​​falling outside this set

[28]. Both methods are used in procedural river generation to supplement the generated values.

# Designed Solution

In the next section of the paper, the solution of the procedural generator of the river path, designed as the part of this work, is presented. The conceptual design of control tiles is explained in more detail.

The role of the control tiles is to create a controllable environment for the river, within which its journey through the 3D world can develop.

The control tile concept introduces two types of points:

* **Control points** - represent places where the river path can continue. If the river cannot continue in a particular direction (for example, because the tile is at the end of the world map), the given control point is removed from tile.
* **Connection points** - connect the path from the control point to all the remaining control points. Thus, they help create a 'bend of the river' as it flows from one control point to another.

Control and connection points are stored in two-dimensional regular grids, and the number of these points varies depending on the type of tile, as shown in Fig. 3, where control tiles of types 3, 5, 9, and 17 are outlined.

The basic feature of control tiles is their ability to respect the positions of control and connection points of their predecessors with a lower density, so that the placement of their own control points does not overlap the previous ones but, on the contrary, complements them, as can be seen in Fig. 4, where tiles of type 3, 5, and 9 are overlapped.

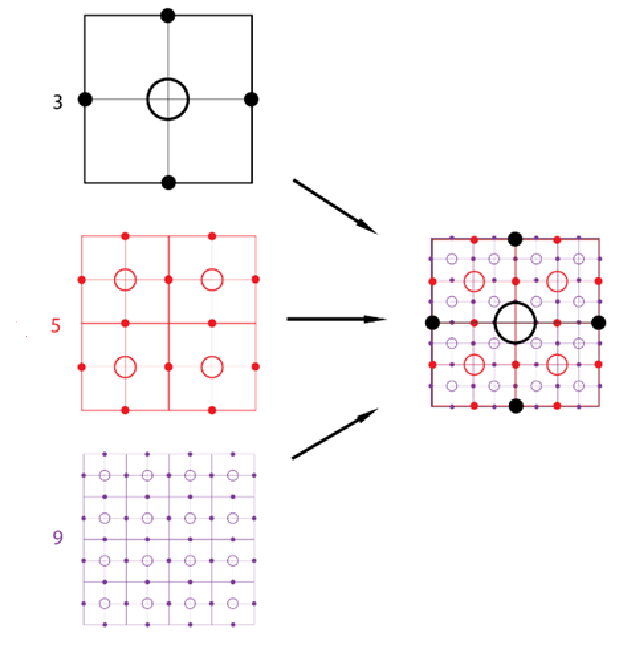
The river cannot pass freely between all types of tiles. In order for final appearance of the river to look natural, the following transition rules were introduced:

1. In the direction of the flow of the river or its branch, only a control tile with the same or higher density of control and connection points can follow a control tile. The goal is that an unnaturally powerful river flow should not follow the river with a narrow channel.

2. Suppression of selected control points based on the side by which the river or its arm entered the tile. The suppression of selected control points regarding the location of the river entrance to the tile is intended to prevent the creation of river arms directed against the original flow of the river flow.

|  |  |
| --- | --- |
|  |  |
| a) | b) |
|  |  |
| c) | d) |

**Fig. 3** Control tiles of the a) type 3, b) type 5, c) type 9, d) type 17, where black circle represents control point and bigger white circle represents connection point.



**Fig. 4** Overlay of control tiles of types 3, 5, and 9, where it can be seen that the connection and control points do not overlap but complement each other.

Creating a river visualization then consists of three separate phases:

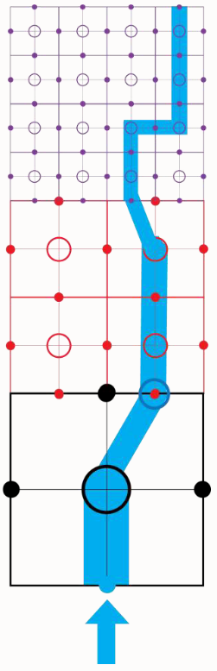
1. Phase is the riverbed sketch phase

2. Phase is the riverbed modification phase

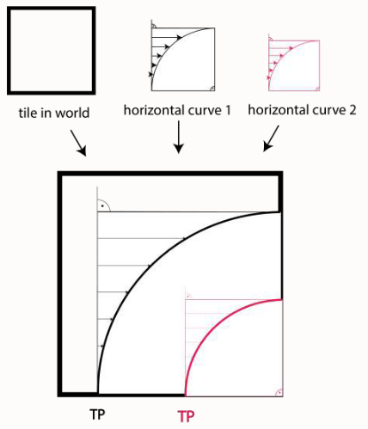
3. Phase is the riverbed rendering phase

The riverbed sketch phase is the phase where the individual control tile types are selected and placed in the space, respecting rule 1 mentioned above.

.



a)



b)



C)

**Fig. 5** Phases of river model visualization a) riverbed sketch phase, b) riverbed modification phase, c) riverbed rendering phase

The riverbed adjustment phase takes into account changes in the direction of the river's flow and creates bends by selecting the corresponding control and connection points within the control tiles.

The riverbed rendering phase represents the final phase in which the shape of the riverbank is definitively determined and rendered in full detail.

The individual stages of creating a visualization of the river can be seen in Fig. 5, visualiations of created river models can be seen in Fig. 6.

# Results

Testing of the proposed river model generator was focused on the speed of the river model generation.

Calculation speed testing was performed on a 64-bit system with an Intel Core i7-8550U microprocessor, with a clock frequency of 1.80GHz and with 8GB of operating memory.

Different measurements were performed for different river model lengths, for each length, ranging from one tile to 100 tiles:

• **Sketch phase speed testing:** The testing measured the duration of feature calculations within the sketch phases.

• **Render phase speed testing:** The testing measured the duration of function calculations from the render phase.

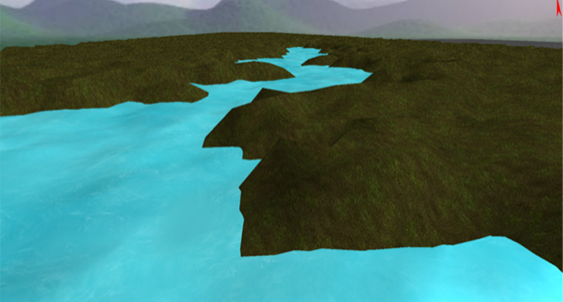
• **Overall generator speed testing:** The testing measured the total duration of calculations performed within the execution of the procedural generator.

Testing in the program took place by recording the start and end times of selected phase calculations. The currentTimeMillis() function from the Java class System was used to record individual times. After finishing the generator calculations, the recorded start and end times for individual measurements were subtracted from each other, and the difference value, indicating the duration of the calculation, was written down.

The duration of calculations for different river lengths (or for different numbers of control tiles representing the river length) was tested. Several measurements were taken for each river length, and the resulting values ​​were subsequently averaged.

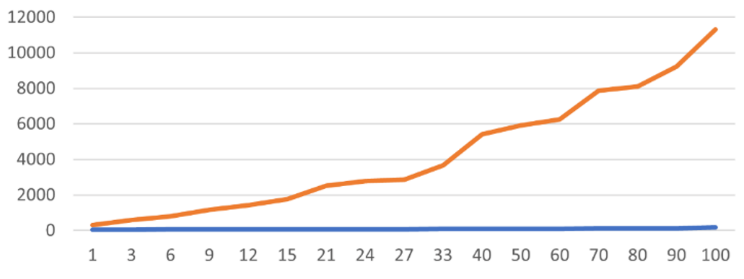
The results of testing the speed of calculations of the procedural generator confirmed the assumption about the effectiveness of using control tiles in terms of speed of calculations.

Speed ​​tests showed that it takes an average of 187 milliseconds to generate a river sketch of 100 control tiles, while it takes 317 milliseconds to generate a final river model of one tile length; thus, the calculation of the sketch phase for the longest measured river model takes on average 41% less time than the calculation of the rendering phase for the shortest measured river model that comprises only one control tile (Fig. 7).



|  |
| --- |
|  |
|  |
|  |
|  |
|  |

**Fig. 6** Examples of visualizations of river models generated using designed procedural content generator.



**Fig. 7** Duration of individual phases of the river model generation using the designed algorithm in milliseconds (ms) - x-axis concerning the size of the river indicated in the number of tiles – y-axis. Blue line represents duration of the sketch phase; orange line represents duration of overall rendering time of river model.

# Conclusion

This paper dealt with the issue of procedural content generation, focusing on it in computer graphics and, more specifically, computer games. It specialized in the procedural generation of a 3D polygonal surface model of a riverbed.

As part of the work, an algorithm was designed and programmatically implemented for generating a riverbed model using control tiles with different numbers of points, which represent control and connection points, thanks to which it is possible to control the direction and shape of the river and its branches. The speed of river model generation was tested, considering that the goal is to build a generator that will allow generating the shape of the river bed in real-time during the construction of the computer game environment.

Our future work in this area will be focused on the practical application of the procedural generation of other aspects of the river. This will include the generation of river bank structures and the river's level, making the river applicable in the open environment of the game. This practical approach ensures that our research has direct relevance and applicability in the field of computer game development.

##### Acknowledgment

This work was supported by KEGA Agency of the Ministry of Education, Science, Research, and Sport of the Slovak Republic under Grant No. 015TUKE-4/2024 Modern Methods and Education Forms in the Cybersecurity Education.

##### References

1. D.S. Ebert, F.K. Musgrave, D. Peachey, K. Perlin, S. Worley, Texturing & modeling: a procedural approach. Morgan Kaufmann, 2003.
2. S. Kranz, E. Fowler, M. Holton, Science & technology in the arts. New York: Van Nostrand Reinhold Co, 1974.
3. T. Short, T. Adams: Procedural generation in game design. CRC Press, 2017.
4. J. Togelius, J. Whitehead, R. Bidarra: Guest editorial: Procedural content generation in games. IEEE Transactions on Computational Intelligence and AI in Games. 2011, Vol. 3, No. 03, pp. 169–171.
5. P. Landin, The Road to Dungeons, Exploring the random art of procedural levels. https://www.minecraft.net/nl-nl/article/the-road-dungeons.
6. G. Smith, El. Gan, A. Othenin-Girard, J. Whitehead, PCG-based game design: Enabling new play experiences through procedural content generation. 2011, doi: 10.1145/2000919.2000926.
7. J. Freiknecht, W. Effelsberg, A survey on the procedural generation of virtual worlds. Multimodal Technologies and Interaction. 2017, Vol. 1, No. 4, pp. 27.
8. J.-D. Génevaux, É. Galin, E. Guérin, A. Prytavie, B. Benes, Terrain generation using procedural models based on hydrology. ACM Transactions on Graphics (TOG). 2013, Vol. 32, No. 4, pp. 1–13.
9. L. B. Leopold, A View of the River. Harvard University Press, 1994.
10. R. Výleta, S. Kohnová, P. Valent, Riešené úlohy z hydrológie I.: Povodie, zrážky, prietok a hydrologická bilancia. Edícia skrípt. Bratislava: Slovenská technická univerzita v Bratislave, 2011.
11. A. Lagae, S. Lefebvre, R. Cook, T. Derose, G. Drettakis, D.S. Ebert, J.P. Lewis, K. Perlin, M. Zwicker, A survey of procedural noise functions. In: Computer Graphics Forum. 2010, pp. 1–20.
12. P. Gonzlaez, J. Lowe, The Book of Shaders. 2015, https://thebookofshaders.com/11/
13. J. Doran, I. Parberry, Controlled procedural terrain generation using software agents. IEEE Transactions on Computational Intelligence and AI in Games. 2010, Vol. 2, No. 2, pp. 111–119.
14. S. Worley, A cellular texture basis function. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. 1996, pp. 291–294.
15. M. Brdička, L. Samek, B. Sopko, Mechanika kontinua. Praha: Academia, 2000.
16. R. Bridson, M. Muller-Fischer, Fluid simulation: Siggraph 2007 course notes video files associated with this course are available from the citation page. In: ACM SIGGRAPH 2007 courses. 2007, pp. 1–81.
17. B. Smyth, Lagrangian and Eulerian descriptions. https://eng.libretexts.org/Bookshelves/Civil\_Engineering/Book%5C%3A\_All\_Things\_Flow\_\_Fluid\_Mechanics\_for\_the\_Natural\_Sciences\_(Smyth)/05%5C%3A\_Fluid\_Kinematics/5.01%5C%3A\_Lagrangian\_and\_Eulerian\_descriptions.
18. R. Bridson, Fluid simulation for computer graphics. AK Peters/CRC Press, 2015.
19. R. Narain, CS 184: Foundations of Computer Graphics. [B.r.].
20. T. Jankuhn, M.A. Olshanskii, A. Reusken, Incompressible fluid problems on embedded surfaces: modeling and variational formulations. Interfaces and Free Boundaries. 2018, Vol. 20, No. 3, pp. 353–377.
21. G. Center, Navier-stokes equations, https://www.grc.nasa.gov/ WWW/k-12/airplane/nseqs.html
22. J.A. Carlson, A. Jaffe, A. Wiles, The millennium prize problems. Citeseer, 2006.
23. F.H. Harlow, J.E. Welsch, Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. The physics of fluids. 1965, Vol. 8, No. 12, pp. 2182–2189.
24. J. Dombroski, T. Gregory, M. Shotande, E. Rackear, Marker-and-Cell Method (MAC), http://plaza.ufl.edu/ebrackear/.
25. Level Set Methods: An Overview and Some Recent Results. Journal of Computational Physics. 2001, Vol. 169, No. 2, pp. 463–502.
26. S. Osher, Level Set Methods. 2003, pp. 3–20.
27. D. Cline, D. Cardon, P.K. Egbert, Fluid flow for the rest of us: Tutorial of the marker and cell method in computer graphics. Brigham Young University. 2013.
28. Harpsers College, Interpolation and Extrapolation. http://dept.harpercollege. edu/chemistry/chm/100/dgodambe/thedisk/labtech/data7.htm.