Generating Layout for Complex Cave-like Levels with Schematic Maps and Cellular Automata

Izabella Antoniuk
Department of Artificial Intelligence, Institute of Information Technology
Warsaw University of Life Sciences – SGGW, Warsaw, Poland.
izabella_antoniuk@sggw.edu.pl

Abstract. In this paper an algorithm for creating cave-like, user-guided layout is presented. In applications such as computer games, underground structures offer unique challenges and interesting space for player actions. Preparation of such areas can be time consuming and tiresome, especially during the design process, when many ideas are often scrapped. Presented approach aims at improving this process. Schematic input is used so the user can quickly define the general layout. Cave system is divided into levels and tiles – easily-parallelizable modules for the following method stages. Cellular automata are used to extend initial system sketch with interesting shapes while the diamond-square algorithm spreads the final terrain heights. Each stage uses the results of the previously performed operations as input, providing space for alterations. Input maps can be reused to obtain different variations of the same system. The final structure is represented as a 3D point cloud. Chosen representation supports multilevel systems and can be used either as a base for further algorithms, or as a final mesh. The presented approach can be easily incorporated into game design process, while visualizing initial layouts and speeding up preparation of unique, interesting and challenging game spaces for the players to traverse.

Key words: Cellular Automata, computer games, Diamond-Square, procedural cave generation, procedural level generation, schematic maps.

1. Introduction

Computer games are growing increasingly more robust, both visually and in terms of the overall complexity. Creating such content can take significant amount of time. While preparing in-game objects manually brings precise and visually appealing results, it can also lead to repeatable content. This is especially the case, when the designer needs to prepare large amounts of similar elements. While tiring for human, appropriately defined algorithms can easily generate such content, achieving required visual complexity without losing the diversity. It is in such applications, that procedural content generation shines the most, speeding up the modelling process, or creating complete elements.

Procedural content generation is not a new area of research. There are quite a few algorithms and approaches addressing this problem. Different solutions can vary in complexity, focusing on single elements, such as plants [36], rivers [26] or roads [19]; generating specific terrain fragments [15, 20, 28], cities [22] or entire worlds [34, 42, 46]. Another division concerns type of usage, focusing either on generating complete content according to user requirements, or on speeding up the modelling process, giving designers another tool to use. Especially in recent years procedural algorithms gained recognition in widely used applications, such as Blender Geometry Nodes [9] or newly announced

Unreal Engine 5.2 Procedural Content Generation Framework [51]. This clearly shows the need for procedural methods in such applications.

Dungeons or caves are very common in computer games. Defining them in terms of algorithm-related parameters can be tricky, due to the existence of multilevel structure with overlapping elements. At the same time, such areas can be the most interesting for players to explore. They are an integral part of numerous games, from classic dungeon crawlers such as Legend of Grimrock [23] and Dungeon Master [13], to more recent productions like The Elder Scrolls V: Skyrim [41], Witcher and Dragon Age series [16,54] or Elden Ring [17]. Defining layout usually requires additional constraints and rules, which makes them more difficult to generate accurately. At the same time, existing solutions usually either focus on 2D maps of such areas [27, 52], or in case of 3D approaches do not provide user with enough control. Generation often requires large amounts of data, producing output that is not easy to modify [10, 11, 12, 14, 31, 39].

Research presented in this paper focuses on developing an algorithm for procedural generation of cave-like structures, that are both visually interesting and complex. Presented method takes into account the multilevel property of the chosen terrain type, and can represent it accurately. User can additionally define the layout of the entire system, using simplified sketches, ensuring that it will follow the initial design, while the algorithm will add detail to it. Finally the method allows both for simplified visualization using point-cloud, as well as further edition of generated content (either using modelling applications such as Blender [9], or by incorporating them into game engines, like Unity [49] or Unreal Engine [50]).

2. Related works

Procedural content generation (PCG) especially in recent years is developing quickly. New approaches are prepared and increasing number of them takes into account specific requirement, coming from fields such as computer games and simulations. At the same time, there are still some drawback that can be seen in those methods. Most of existing solutions can be roughly divided into two main categories: complex generation or methods focused on speeding up the modelling process.

Both approaches can be interesting for applications such as computer games, assuming certain requirements are met. Designers would usually want to transfer their vision into the final outcome of the algorithm. Because of that it is important to include some way to define content properties. At the same time numerous, unintuitive parameters can have exactly opposite effect, over-complicating the process and making it to tiresome. Existing methods use various types of input files, as well as ways to evaluate generated elements. Most interesting from the point of view of research performed in this paper, are solutions that:

- use different types of maps as algorithm input,

- focus on content meant for computer games, or
- generate cave-like and dungeon-like structures.

All of the above groups represent some properties of the presented approach. At the same time no method was found meeting all of them from the chosen field.

For a comprehensive survey of various PCG methods used to create virtual world elements see [44, 53, 56].

2.1. Generating world elements using input maps

One of the major areas, where procedural content generation shines, is creating world elements. Producing such areas (similar in structure, but not repeatable) using algorithms is more than justified. At the same time defining terrain with input maps makes it easier for the user to make sure that the results meet his requirements.

A complex approach with multiple maps representing information about terrain details is presented in [47]. Authors use separate files to define general terrain height, bodies of water, vegetation, roads and buildings, and later combine them to represent full scene. The method can model interactions between different layers, i.e. creating bridge if a road crosses over river. This approach was further developed in [43, 45, 46], adding detail to the method and improving the generation process and procedures used to connect different elements.

Slightly different approach, instead of using schematic maps to outline terrain, applies them as modifiers [55]. In that case provided sketch map defines key points, such as canyons, rivers or mountain ranges. This map is then used to generated 3D scene, using USGS DEM information for additional details.

A series of solutions focuses on generating different virtual world fragments using simplified inputs [2, 3, 4, 5, 6, 7]. Similarly to research presented in this paper, terrain is divided into tiles, and – in case of underground structures – levels, while the generation process is defined by user-set properties. The approach also takes into account various constraints related to the computer games in general. Final results are represented as editable, 3D models.

As can be seen in different approaches, even simple input maps can be used to generate complicated and engaging content, with some of them using just a single map for that purpose [1, 38]. Such input is easier for a designer to understand, than sets of numeric parameters. High level of control with intuitive file structure and its influence over final result are the key elements that decided the types of maps used in the approach presented in this paper.

2.2. Procedural generation for games

When it comes to computer games, any content meant for such application needs to have specific properties. Preparing algorithms that take various requirements into account is another vast area of research. A comprehensive survey of PCG methods used in computer games was presented in [25].

As was noted in [21], there are quite a few areas, where repeatability of different elements can be noted. Authors point out, that since repetition is not a natural occurrence, with real wold containing infinite numbers of unique patterns, it can often result in player losing the immersion because of this. Repetitive elements can tire the player. When recognized patterns can additionally be transferred to gameplay strategies, the difficulty will also decrease, resulting in boredom. Making sure it is not the case for newly created content is an excellent task for procedural generation algorithms.

In [8] a fitness function is used to evolve mazes that meet specified requirements. Layouts are produced by adjusting parameters, ensuring that both ends of the maze are always connected. Different evolution-based approach focuses on generating 3D terrain fragments, that the user can adjust [37]. In this case, the terrain is constructed based on selected patches, creating a seamless crossover, in theory closer to the user requirements.

Another set of examples instead of defining properties of the created world, takes into account the story that will happen in it. In [24] authors use story, to later generate world supporting its key elements. This solution is capable of creating complex terrains, that are well adapted to given input. While the maps are two-dimensional, they could be used as a base for further work in 3D space. In [32] authors incorporate user-defined key points, and relationships between them. This allows the creation of a map with strategically placed towns and cities. Content created in such a way aligns with specifically set constraints, consistent with the game story.

In case of computer games, level of control is equally important as interesting results. In [30] the user can choose a set of actions, that will later be represented in the resulting map. Used constraints are all gameplay-related and need to be specified by the designers. The operation of the algorithm was presented on the example of Dwarf Quest game, and resulted in complicated layouts.

One interesting method describes procedural level generation using snappable meshes [40]. Authors use set of predefined assets with established connection points, a set of constraints describing how they can be connected and general way the level will be constructed. Different level types can be created and implementation in the Unity game engine was also prepared. Although the main focus of the algorithm is to "avoid size and layout limitations", it is heavily dependent on the quality of the prepared assets. The map generated can contain multiple levels, but this again depends from the structure of initial assets.

2.3. Creating cave and dungeon structures

When it comes to procedural generation of underground structures, a series of additional constraints need to be considered. The layout tends to be more complex than in case of the surface areas. Additionally such structures in real world tend to have multiple levels with overlapping areas, that cannot be represented by a simple height map.

One of many approaches to cave generation in particular considers the problem of creating natural-looking structures, with main focus put on the karst caves [18]. The method is implemented in Unity, and prepares both the layout of the cave system, along with individual shape of passages, textures and cave features (such as speleothems). The generation is heavily based on the natural process of cave formation, although it is simplified to expedite the computation. Unfortunately it does not provide any way to define or adjust the layout of the cave system. It also doesn't take into account any computer game requirements apart from generation time. At the same time it is noted, that current version of the method is not adjusted to such application.

Another intersecting example uses genetic algorithms to evolve dungeons according to user specifications [29]. Authors use two maps for this process. High-level sketch of the dungeon is used to define overall connectivity of different fragments and the content of those which are passable. Second map is a low-level, high resolution representation of individually generated segments. In the second evolution step individual segments are generated. Authors take into account many game-specific requirements. Generated shapes can be complex and visually interesting. The resulting map is mostly two-dimensional though, without any vertical transitions.

Algorithm presented in [31] focuses on generating 3D caves for application in computer games. Method consists of two main steps. First one uses L-system to define the structural points – the general layout of the created level. Tunnels and caves are generated after that, by wrapping a meta-ball along paths defined in the first step. While initially voxel representation is used, final terrain is obtained by converting resulting scene to mesh with assigned textures and shading. The method can create complex, multilevel structures with various features. Unfortunately, user has very little control over resulting layout, making it difficult to apply in computer games where specific terrain properties are required.

Overall, while there are quite a few interesting approaches breaching the subjects presented in this paper, none of them meets all of the defined requirements. Prepared algorithm builds on those drawbacks, using input maps to ensure user control and allowing definition of multilevel structures with precise layout. Generation process can be influenced during various stages, with final results stored in an easily editable manner.

3. Input maps

One of the more important aspects of procedural generation for computer games, is the definition of initial input files. Using only generation parameters poses some problems. While such approach can be sufficient to simple applications, computer designers usually will require more significant way to define final content. Another problem is that parameters in general tend to be harder to understand in terms of their influence over the final object. At the same time, input which is to specific will reduce procedural generation

to the slightly faster manual modelling (as is the case with SpeedTree application [48]). While using such tools tends to speed up the level creation it also doesn't work well with large amounts of repeatable elements. The need to create large quantities of similar content often results in areas that are visually similar to each other. Procedural content generation can shine especially in such areas, assuming it can at the same time sufficiently include input from the human designer.

To achieve high level of control without reducing the generation process to tedious, manual modelling, set of schematic input maps is used. In presented approach user needs to prepare total of two maps, to represent the overall structure of underground system: placement map, and system layout map.

3.1. Placement map

First used map defines placement of individual tiles inside each level vertically. Standard approach for the height maps is used, with values represented in grey-scale in range (0, 255). The main difference lies in how those values are applied. In that case each pixel represents single region in final terrain (tile). Height value represents placement of tile in single level. Actual height values for tiles are scaled according to each level spread (with single step value resulting from dividing overall level height by 256). Each value in the placement map corresponds directly to basic height of the tile, used for further operations. Since placement map does not include tile size, once defined it can be reused for systems with the same structure, but different sizes of the individual elements, making it easy to experiment with various levels of complexity for the final system. Example visualization of tile spread with corresponding height map is presented in Figure 1.

3.2. System layout map

Second map represents general system layout. Since the designer might want to define key features (i.e. large room in chosen location, or some crossings that must occur), this map is used to represent such elements. There are few key aspects that need to be represented:

- definition of the general layout for the underground system,
- indication of existing connections between tiles,
- exclusion of tile connections when necessary,
- indication of passages leading to lower levels.

In [5, 6] similar approach was used, with one map defining tile placement, and two additional ones denoting connections and type of terrain in each region. While specific, such definition is not intuitive enough, especially in terms of defining connections.

To avoid such problems, in the presented approach all key elements are represented by using different coloured annotations on single image. User can sketch a simple map, where white colour represents the general system layout, that will be later used during

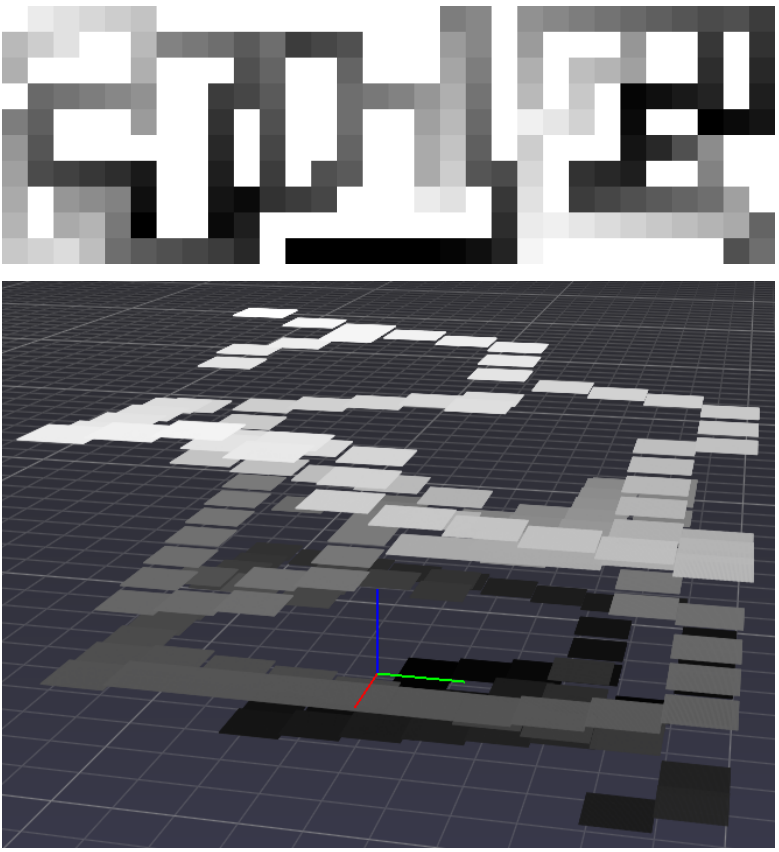


Fig. 1. Placement map used for deciding initial tile location (top), with corresponding point-cloud visualization representing a region spread in 3D space (bottom). The placement map is prepared for 3-level dungeon, where each level contains 10×10 separate tiles, with tile size equal to 51. The tiles were marked in grayscale, to represent the overall system structure, with lighter values representing higher regions.

generation. At the tile edges, connections to lower levels can be specified using green colour, and passage exclusions can be defined with blue. Tile connections are obtained automatically from general system shape (transitions for each region are named according to the tile they lead to: either Top, Bottom, Right or Left). There are four cases that need to be considered:

- Tile edge has green mark, indicating connection to lower level. The connection will be saved with 'L_' prefix and will be used in level connection stage during generation process.

- Tile edge has blue mark, indicating excluded connection. This direction will be saved with 'E_' prefix and even if the system shape will reach the edge of the tile, it will not be considered as connection in future operations.
- Tile edge has no marks and system sketch (white) reaches edge of the tile. The connection will be saved in basic form (Top, Bottom, Right or Left), and will be considered during generation process.
- Tile edge has no marks, and system sketch does not reach edge of the tile. The connection will be saved as 'None', and will not be considered during generation process.

Example system map is presented in Figure 2.

Additional problem with underground systems concerns overall space representation. For the created structure to be interesting in terms of exploration it should contain multilevel and complex structures. Standard 2D files are not able to directly represent such areas. To address that problem, the space in generated caves is divided into vertically aligned levels, with structures that are not overlapping with each other. Each level has size and spread. The size of level defines number of tiles along each side. Spread decides single level height and is a base for division used while placing individual regions. Vertical transitions can be realised by connecting tiles between different levels, allowing creation of more complex structures. All input maps contain combined information for all levels in created system, with highest level data placed on the left side, and succeeding, lower levels placed next to it (see Figures 1 and 2).

4. Layout generation

After user prepares the input files, initial data is derived from them and used to generate the underground system. The process is divided into three, separate stages: preparing initial system layout, generating heights for cave shape in each tile, and combining data.

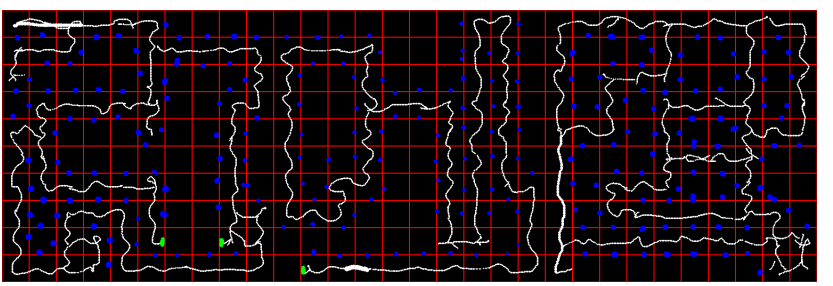


Fig. 2. System layout map used for shape generation. Presented sketch is prepared for 3-level dungeon, where each level contains 10x10 separate tiles. Single tile size is set at 51x51 pixels. Red-coloured grid marks borders between individual tiles.

First stage generates the overall system shape in each tile, generating cave-like shapes using cellular automata algorithm, and connecting it to user-prepared system sketch. This map can be later extended using individual operations, either on entire system or on individual tiles. After the initial map is prepared, heights for each pixel representing system shape are generated using the diamond-square algorithm, to add 3D details. The data from both stages is than combined, to fully reprsent the final system shape. Each pixel in the original map correscponds directly to single vertex in 3D space. Results are visualized as a point-cloud. Pseudocode outlining the entire procedure is presented in Algorithm 1.

4.1. Initial system shapes with cellular automata

First step during the generation process concerns creating the initial system shape. Taking into account the connections between tiles, the system sketch created by the user will be expanded using cellular automata algorithm.

Such approach was chosen for several reasons. Firstly, cellular automata can produce realistic results, that resemble the real-world structures. In [27] this algorithm produced visually interesting shapes. At the same time, it is hard to control the layout of the system without some modifications. One huge drawback is the connectivity of the generated structures. Especially in applications such as computer games, existence of areas that

Algorithm 1 Cave system layout generation from schematic input maps.

```
Input: height map, system sketch
for tiles in system do
    Generate cellular automata shapes
    Connect shapes to sketch (system sketch)
end for
System map = generated map
for User input action do
    updated system map = Perform user action (System map)
    System map = updated system map
end for
for tiles in system do
    Check user options for height generation
    Generate heights in tiles (System map, user options)
end for
Final system = Combine data (System map, height map, system sketch)
Calculate point coordinates for export (System map, height map, system sketch,
Final system)
```

are not connected to the main system can be problematic. It is even more important if additional method is used to place key objects or the player himself as it can create a scenario, where either the player is stuck in small area, or the key elements required for progression are unreachable. Both situations are extremely undesirable. Additionally, designer would usually want the cave system to follow certain layout. By expanding the sketch, instead of generating the entire structure, this requirements will be easily met, while producing visually interesting and complex shapes.

The chosen approach uses cellular automata algorithm, to generate cave-like shapes of various sizes across the tile. Those shapes are then attached, firstly to the initial sketch presented by the user, and later to the overall structure created during generation. With multiple operations used, the final system can grow without the danger of disconnected elements being created along the way.

During this stage the initial map of the system is created. While complex and interesting, in order to provide greater level of control over system shape, additional operations were required. After the initial generation, user is provided with additional methods for further edition of the cave system shape (either on single tile, or on the system as a whole). The resulting image can also be modified in external application.

For the edition stage (apart from manually drawing over created system), few basic operations are available, so the final shape will better meet the user requirements. Individual methods work as follows.

- Run CA – performs single run of cellular automata (CA) algorithm for the current system in the classical form.
- Combine with sketch – removes all disconnected elements, leaving only those, that are connected to the main system; method was added because methods "Remove CA shapes" and "Thin system edge" can produce unattached shapes.
- Add CA shapes – generates new CA shapes and connects them to the existing system.
- Remove CA shapes – generates new CA shapes and removes them from the existing system.
- Fill system edge – runs along the edge of existing system (edge is defined, when a cell has at least one neighbour, that is not classified as system shape or white), and expands the shape by drawing circle at each edge cell.
- Thin system edge – runs along the edge of existing system and thins it by changing the cell values to wall (black).

Results for single run of each method are presented in Figure 3.

Shape generation using cellular automata are the longest part of the presented approach. Because of that, all of the operations can be done either on all the tiles in the system or on individual regions. This will also address the situation, when user likes the general look of the structure, but wants to improve its individual fragments, without regenerating everything. Example of full system shape obtained during this stage of generation is presented in Figure 4.

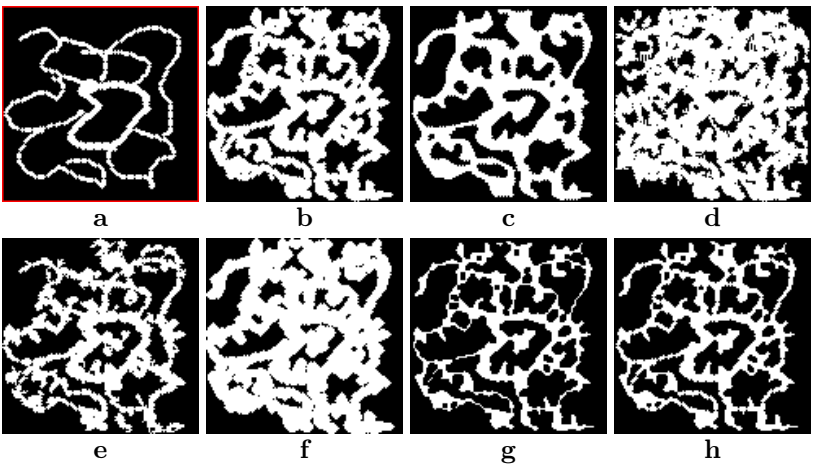


Fig. 3. Results of individual operations on single tile. (**a**) First two images show original system sketch, (**b**) additionally outlined in red and initially generated shape. The following images present results of each operation on (b); respectively: (**c**) Run CA, (**d**) Add CA shapes, (**e**) Remove CA shapes), (**f**) Fill system edge and (**g**) Thin system edge. Final image (**h**) shows result of "Combine with sketch operation", performed on the shape presented in (g).

Fig. 4. Initial system shape generated from input maps presented in Figures 1 (top) and 2. Single tile size is set at 51×51 pixels.

4.2. Diamond-square height generation

After the initial generation, each tile contains flat representation of the system. Next step involves generation of height map, to include in that shape. Initial placement map will spread the tiles vertically, while this step will add required details to each area. The diamond-square algorithm is used in that aspect. This method was chosen, since the general direction of the height map can be steered, by changing the initial values placed in the corners.

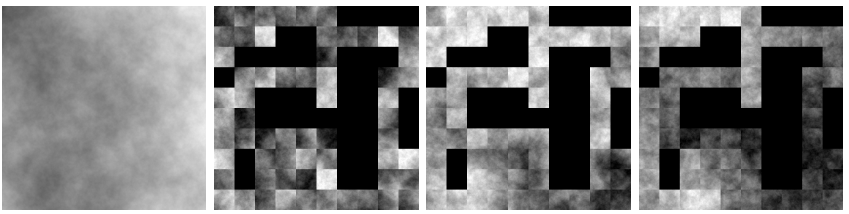


Fig. 5. Height maps generated using the diamond-square algorithm with different corner settings: (**a**) for the entire level without tile division; for single tiles, with initial corner values: (**b**) generated randomly, (**c**) calculated from the initial tile placement for three tiles adjacent to each corner, and (**d**) obtained from the two tiles adjacent to the corner for the edges with connections.

Since depending from the final application, different properties of the terrain might be necessary, few variations of the height map generation algorithm are considered:

1. for the entire level, for each of the defined levels;
2. separately for each tile;
3. for each tile, with initial values for the corners calculated from the placement of the neighbouring tiles;
4. for each tile, with initial values for the corners calculated from placement of neighbouring tiles, along connected edges.

For the first two cases the initial heights of the corners are generated randomly, the only difference being the map size. Diamond-square method requires the size of image equal to $2^n + 1$. Firstly the appropriate size is obtained, by finding smallest possible value, that will allow creation of either single tile, or entire level. The corner values are then randomly generated and the algorithm proceeds, with final map selected from subset of the generated one.

In the third case, initial values are calculated by averaging height of neighbouring tiles connected to each corner. The method takes initial tile heights obtained from placement map, and average is calculated according to the number of elements in the final set. For the corners, that have no neighbours, random height value will be taken.

Final method takes into account connections between tiles. In that case, the method first checks if the connection exists along each of the two edges connected to the current corner. Only if such connection exists, the neighbouring tile height is added to the average. This method was used to better reflect the initial structure of the cave system obtained from the input maps. Example height maps generated for each version of the diamond-square algorithm are presented in Figure 5.

4.3. Point-cloud visualization

Final step of the generation process is the visualization in 3D space, using Python Point Processing Toolkit library (pptk) [35].

During first step of this procedure, heights obtained from the diamond-square algorithm are used to update placement of each point according to those values. Single point in the final representation corresponds directly to individual pixels in tiles. Point Processing Toolkit requires a list containing 3D coordinates, so firstly the algorithm makes sure, that the value from height map is transferred to tile points accordingly. With this initial operation done, the algorithm then focuses on connecting the tiles and preparing final list of points for visualization.

Since the tiles are initially placed in 3D space using the data from placement map, transferring the position from the generated height map would result in gaps between system fragments. Therefore, before visualization, the point height values along the edges are updated in each tile to correct that problem. At the same time values are transferred to the format required by the used library. The points at the edges of the tile are first tilted and then connected vertically, making sure that each transition will be included into final system. Since number of connections to the lower levels usually is much lower, than the total number of connections, those transitions are extracted into separate list and iterated through after the initial process is finished. The outline of the entire operation is presented in Algorithm 2.

Last step takes all the final heights obtained so far and visualizes them. Firstly, all points that are classified as wall (black value on the images) are excluded. Resulting data structure will contain all points organized in a list, including final placement of each point in 3D coordinates. Points are then visualized in pptk tool, using 'gray' colour map – as a result higher points have whiter values in the gray-scale range, better showing the structure of the entire system.

Depending from the generation type chosen for the diamond-square algorithm, the effects of the final visualization will vary, since the heights obtained from that algorithm are directly transferred to the final point set. Overall height is scaled to the appropriate range, obtained from level spread.

Algorithm 2 Point coordinates preparation for the visualization process

```
for tiles in system do
    Check tile connections
    for connections in tile do
        Tilt tile along connected edge.
        Connect tile to the neighbour
    end for
end for
for connection in level connections do
    Connect tile to the neighbour in lower level.
end for
```

5. Results and discussion

In order to evaluate algorithms presented in this paper, method implementation was prepared using Python programming language. All tests were performed on a personal computer with Intel® Core™ i7-9750H CPU 2.60GHz, with Nvidia GeForce RTX 2070 graphics card and 32GB of RAM.

First part of evaluation was mainly visual in nature, comparing generated shapes to natural caves, and evaluating potential game requirements in that aspect. Cave structures in real world usually will contain spaces with characteristic shapes, big enough for a person to enter. Layout can be quite complicated, often with multi-layer structure and differently shaped passages of various length, height and general structure. It is caused by characteristics of the formation process, where acidic water first dissolves the rock, to wash it out later on [33]. Depending on the types of rocks and their strengths in different places, large structures can be placed next to the tight passages with various formations. All of those elements are important from gameplay point of view, providing additional challenge for the player. Initial structure containing such elements can be achieved with the system sketch. Larger areas can be easily defined by the designer, as well as any type of desirable layout. At the same time, while the generated shapes follow the sketch, they are not repetitive, and provide interesting areas to traverse, similar in their structure to the natural caves (see Figure 2, and the resulting map in Figure 4).

Another set of parameters concerned the cave-like structures presented in various computer games. In general few key features of used cave systems can be defined:

- system shape should be controllable;
- the system must contain only connected areas;
- system should contain side-spaces where potential enemies or in-game objects can be hidden;
- system should contain narrow passages, that can be blocked by enemies;
- system layout should be complex enough, that it poses challenge for the player, but at the same time is possible to represent on 2D map.

Underground structures generated by the presented method have all of the above features. Thanks to the used input maps, even very complex layouts can be represented, ensuring that the designers idea will transfer to the resulting structure. At the same time it can be edited, both during the generation, as well as after it, in other applications. Algorithms incorporated in the process can create natural-looking systems, with numerous niches and narrow passages placed along user-defined sketch. It is also ensured, that any areas in the system will be connected to the main passage, so there is no problem with inaccessible spaces. Examples of multiple maps obtained from single sketch are presented in Figure 6. As can be seen, depending from initial user outline, resulting system can contain complex structures with various features, while retaining

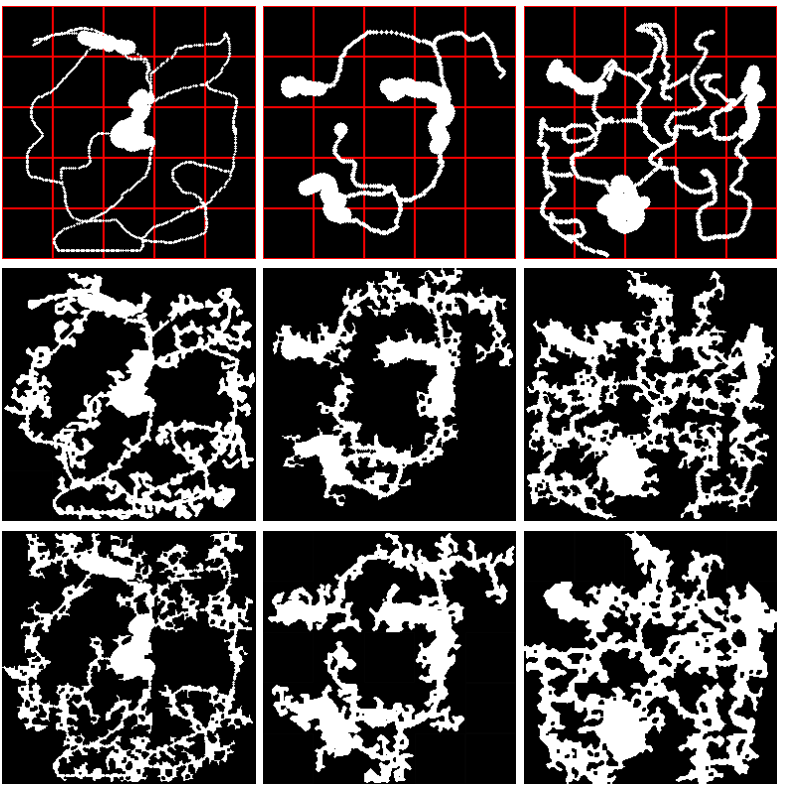


Fig. 6. Examples of used maps (top row) and two variations of resulting cave systems (remaining rows). The map consisted of single level 5×5 tiles, with tile size set at 51 pixels. The generation of each variation, including additional operations (like thinning/filling system, adding CA to tiles, etc.) took under one minute.

key design elements. At the same time, different variations of the same system can be easily produced.

Remaining tests concerned the overall method performance. Since presented approach is mainly focused on speeding up the designers workflow it needs to compute relatively fast. While some waiting time is acceptable, it should also be short enough to provide user with mostly real-time interaction. If the designer needs to wait long hours for the result, only to decide that it is not satisfactory, such approach will not be sufficient. Figure 7 presents generation times for consecutive steps of used method, under various conditions.

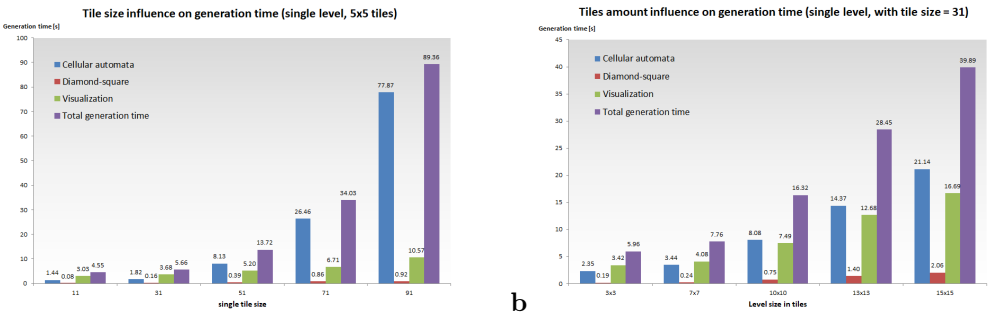


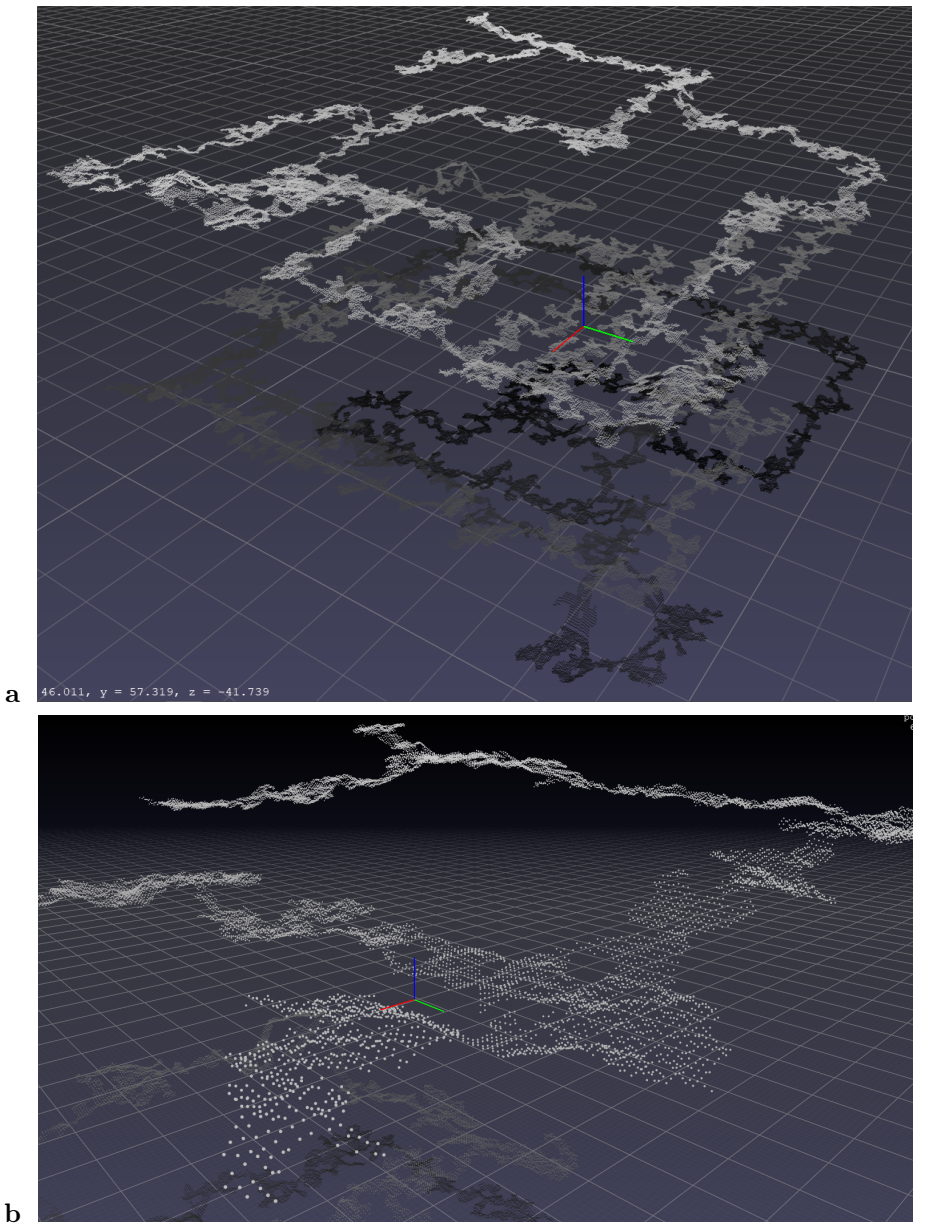
Fig. 7. Generation times obtained for prepared methods. (**a**) Key steps of prepared algorithm are evaluated for increasing tile size, and (**b**) number of tiles in level. Each result was averaged from data obtained during 100 iterations.

In general, increasing tile size has greater impact on the generation time than higher number of tiles. This is mostly due to used multiprocessing method, where each region is generated separately. It can be seen, that for larger tiles, the main factor during generation was the cellular automata algorithm, used to obtain system shape. This changes with larger number of tiles, where CA algorithm and visualization have closer values. Since the visualization part can be omitted, i.e. when creating data for other applications, this might be a better solution in those cases.

Final tests checked the influence the type of height-map in the diamond-square algorithm has on generation time. This test was done for single level, 5×5 tiles, with tile size set at 91 pixels. Results for the tile based methods were very similar, reaching 1.01, 1.01, and 0.97 s, for height generation methods 2, 3 and 4, respectively (see Section 4.2). Since in all cases only the corner values are changed, it is only natural, that the differences are minimal. The only real influence lies with the method that generates single height-map for the entire level (method 1), and this is again, due to the used multiprocessing method, and larger size of single item in this case.

Overall, obtained times are more than satisfactory. The method works close to real time, and even larger levels can be generated relatively quickly, depending from their structure (i.e. single level, 15×15 tiles with size equal to 31 takes less than 40 seconds to generate and visualize). While not fully real-time, it still provides user with possibility to obtain different variations for the chosen layout at reasonable intervals. At the same time, the generation is fast enough, that the system can be edited and improved close to real time – especially when user wants to change few tiles, instead of the entire system. Example visualization of fully generated, three-level cave system, with a close-up of one of its fragments is presented in Figure 8.

While presented algorithm in its current form has large potential, there are still few areas for improvements, that can be addressed in future work. Firstly, visualization in applications such as Blender, Unity or Unreal Engine might bring additional insight in



a

b

Fig. 8. Example point-cloud visualization of the system (**a**) after the full generation process, (**b**) with close-up of system fragment. The system was generated from input maps from Figs. 1 and 2.

terms of general requirements for used methods. 3D modelling environments and game engines have different specificity, and might require additional changes to the algorithm stages. Creating a method for more interactive visualization might also be interesting, i.e. in the Unreal Engine. Another potential area concerns time improvement for larger tiles. While even very complex systems can be represented with smaller elements, some user might still want to use larger areas. Finally, further edition of point cloud generated by the method would be required. Creating mesh, or using prefabs to build a game level based on generated points is an interesting, and most logical next step in that aspect.

Overall, the presented approach is very promising, with various possibilities for development. It can be adapted to different games, and speeds up the design process. At the same time, with minimal additions it can generate full terrain for a simple game.

6. Conclusion

In this paper a procedural layout generation algorithm for multi-level, complex, cave-like systems was presented. Described method uses two schematic maps, one for placing tiles inside each level, and one for defining general layout and system transitions. Results can be represented and visualized in 3D space.

Since the main method application is focused on computer games, different properties were taken into account. The designer can not only define the layout of the entire system, but also correct obtained results at different stages of the generation. The method works at an interactive rate, producing complex layouts in reasonable time (i.e. generating and visualizing system with single level 15×15 tiles with size equal to 31×31 takes 39.89 s). With all those elements taken into account, it can be used either for terrain generation in simple game, or during design process to quickly visualize different layouts. Use of schematic maps not only provides designer with a method to quickly and intuitively define desired results, but also allows fast creation of variations that share the same transitions. Since the results are presented as a point-cloud, it can be further edited and modified, either by transferring those points into 3D modelling environment, or creating levels using other methods, i.e., with predefined assets.

References

[1] D. B. Adams. *Feature-based Interactive Terrain Sketching*. Master's thesis, Brigham Young University, 2009. https://www.proquest.com/openview/ccd517e9097577329a8faf0d38327518.

[2] I. Antoniuk, P. Hoser, and D. Strzęciwilk. L-system application to procedural generation of room shapes for 3D dungeon creation in computer games. In: *Proc. Int. Multi-Conf. Advanced Computer Systems*, pp. 375–386. Springer, 2018. doi:10.1007/978-3-030-03314-9_32.

[3] I. Antoniuk and P. Rokita. Feature-based procedural generation of adjustable game content. *Challenges of Modern Technology*, 5(4):21–26, 2014. https://www.infona.pl/resource/bwmeta1.element.baztech-bd463454-b467-4eed-b4a7-a4f9b74a9d99.

[4] I. Antoniuk and P. Rokita. Procedural generation of adjustable terrain for application in computer games using 2D maps. In: *Pattern Recognition and Machine Intelligence: Proc. 6th Int. Conf. PReMI 2015*, vol. 9124 of *Lecture Notes in Computer Science*, pp. 75–84. Springer, Warsaw, Poland, Jun 30-Jul 2015. Article No. 10. doi:10.1007/978-3-319-19941-2_8.

[5] I. Antoniuk and P. Rokita. Generation of complex underground systems for application in computer games with schematic maps and L-systems. In: *Proc. Int. Conf. Computer Vision and Graphics*, pp. 3–16. Springer, 2016. doi:10.1007/978-3-319-46418-3_1.

[6] I. Antoniuk and P. Rokita. Procedural generation of underground systems with terrain features using schematic maps and L-systems. *Challenges of Modern Technology*, 7(3):8–15, 2016. doi:10.5604/01.3001.0009.5443.

[7] I. Antoniuk and P. Rokita. Procedural generation of multiclevel dungeons for application in computer games using schematic maps and L-system. In: *Intelligent Methods and Big Data in Industrial Applications*, pp. 261–275. Springer, 2019. doi:10.1007/978-3-319-77604-0_19.

[8] D. Ashlock, C. Lee, and C. McGuinness. Search-based procedural generation of maze-like levels. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):260–273, 2011. doi:10.1109/TCIAIG.2011.2138707.

[9] Blender Reference Manual, version 3.5. `https://docs.blender.org/manual/en/latest/index.html`, Accessed: 22.04.2023.

[10] M. Boggus and R. Crawfis. Explicit generation of 3D models of solution caves for virtual environments. In: *Proc. 2009 Int. Conf. Computer Graphics & Virtual Reality (CGVR)*, pp. 85–90. Las Vegas, Nevada, USA, 2009. `http://web.cse.ohio-state.edu/tech-report/2009/TR18.pdf`.

[11] M. Boggus and R. Crawfis. Procedural creation of 3D solution cave models. In: *Proc. 20th IASTED Int. Conf. Modelling and Simulation*, pp. 180–186. 6-8 Jul, Banff, Alberta, Canada 2009. `https://www.actapress.com/Abstract.aspx?paperId=35085`.

[12] M. Boggus and R. Crawfis. Prismfields: A framework for interactive modeling of three dimensional caves. In: *Proc. Int. Symp. Visual Comput.*, pp. 213–221, 2010. doi:10.1007/978-3-642-17274-8_21.

[13] J. Calderon. Dungeon Master. `https://ka-plus.pl/en/dungeon-master/`, Accessed: 18.12.2023.

[14] J. Cui, Y.-W. Chow, and M. Zhang. Procedural generation of 3D cave models with stalactites and stalagmites. *IJCSNS*, 11(8):94, 2011. `https://ro.uow.edu.au/infopapers/3625/`.

[15] D. M. De Carli, C. T. Pozzer, F. Bevilacqua, and V. Schetinger. Procedural generation of 3D canyons. In: *Proc. 2014 27th SIBGRAPI Conf. Graphics, Patterns and Images*, pp. 103–110. IEEE, Rio de Janeiro, Brazil, 26-30 Aug 2014. doi:10.1109/SIBGRAPI.2014.41.

[16] Dragon Age game series webpage. `https://www.ea.com/en-gb/games/dragon-age`, Accessed: 18.12.2023.

[17] Elden Ring game webpage. `https://en.bandainamcoent.eu/elden-ring/elden-ring`, Accessed: 18.12.2023.

[18] K. Franke and H. Müller. Procedural generation of 3D karst caves with speleothems. *Computers & Graphics*, 102:533–545, 2022. doi:10.1016/j.cag.2021.10.002.

[19] E. Galin, A. Peytavie, N. Maréchal, and E. Guérin. Procedural generation of roads. *Computer Graphics Forum*, 29(2):429–438, 2010. doi:10.1111/j.1467-8659.2009.01612.x.

[20] M. N. Gamito and F. K. Musgrave. Procedural landscapes with overhangs. In: *Proc. 10th Portuguese Computer Graphics Meeting*, vol. 2. Lisbon, Porugal, 2001.

[21] S. Greuter and A. Nash. Game asset repetition. In: *Proc. 2014 Conf. Interactive Entertainment*, pp. 1–5, 2014. doi:10.1145/2677758.2677782.

[22] S. Greuter, J. Parker, N. Stewart, and G. Leach. Real-time procedural generation of 'pseudo infinite' cities. In: *GRAPHITE '03: Proc. 1st Int. Conf. Computer graphics and interactive techniques in Australasia and South East Asia*, pp. 87–ff. ACM, Melbourne, Australia, 11-14 Feb 2003. doi:10.1145/604471.604490.

[23] Legend of Grimrock game webpage. `https://www.grimrock.net/`, Accessed: 18.12.2023.

[24] K. Hartsook, A. Zook, S. Das, and M. O. Riedl. Toward supporting stories with procedurally generated game worlds. In: *Proc. 2011 IEEE Conf. Computational Intelligence and Games (CIG'11)*, pp. 297–304. IEEE, 2011. doi:10.1109/CIG.2011.6032020.

[25] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1–22, 2013. doi:10.1145/2422956.2422957.

[26] R. Huijser, J. Dobbe, W. F. Bronsvoort, and R. Bidarra. Procedural natural systems for game level design. In: *Proc. 2010 Brazilian Symp. Games and Digital Entertainment*, pp. 189–198. IEEE, Florianopolis, Brazil, 08-10 Nov 2010. doi:10.1109/SBGAMES.2010.31.

[27] L. Johnson, G. N. Yannakakis, and J. Togelius. Cellular automata for real-time generation of infinite cave levels. In: *PCGames '10: Proc. 2010 Workshop on Procedural Content Generation in Games*, pp. 1–4. Monterey, California, 18 Jun 2010. Article No. 10. doi:10.1145/1814256.1814266.

[28] K. R. Kamal and M. Kaykobad. Generation of mountain ranges by modifying a controlled terrain generation approach. In: *Proc. 2008 11th Int. Conf. Computer and Information Technology*, pp. 527–532. IEEE, Khulna, Bangladesh, 24-27 Dec 2008. doi:10.1109/ICCITECHN.2008.4803058.

[29] A. Liapis. Multi-segment evolution of dungeon game levels. In: *Proc. Genetic and Evolutionary Computation Conference*, pp. 203–210, 2017. doi:10.1145/3071178.3071180.

[30] R. Van der Linden, R. Lopes, and R. Bidarra. Designing procedurally generated levels. In: *Proc. Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013. doi:10.1609/aiide.v9i3.12592.

[31] B. Mark, T. Berechet, T. Mahlmann, and J. Togelius. Procedural generation of 3D caves for games on the GPU, 2015. `https://portal.research.lu.se/en/publications/procedural-generation-of-3d-caves-for-games-on-the-gpu`, Paper presented at *Foundations of Digital Games*.

[32] E. A. Matthews and B. A. Malloy. Procedural generation of story-driven maps. In: *Proc. 2011 16th Int. Conf. Computer Games (CGAMES)*, pp. 107–112. IEEE, 2011. doi:10.1109/CGAMES.2011.6000324.

[33] A. N. Palmer. Origin and morphology of limestone caves. *Geological Society of America Bulletin*, 103(1):1–21, 1991. doi:10.1130/0016-7606(1991)103<0001:OAMOLC>2.3.CO;2.

[34] A. Peytavie, E. Galin, J. Grosjean, and S. Mérillou. Arches: A framework for modeling complex terrains. *Computer Graphics Forum*, 28(2):457–467, 2009. doi:10.1111/j.1467-8659.2009.01385.x.

[35] Documentation for Point Processing Toolkit (pptk) Python library. `https://heremaps.github.io/pptk/index.html`, Accessed: 22.04.2023.

[36] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer Science & Business Media, 2012. doi:10.1007/978-1-4613-8476-2.

[37] W. L. Raffe, F. Zambetta, and X. Li. Evolving patch-based terrains for use in video games. In: *Proc. 13th Annual Conf. Genetic and Evolutionary Computation*, pp. 363–370, 2011. doi:10.1145/2001576.2001627.

[38] T. Roden and I. Parberry. From artistry to automation: A structured methodology for procedural content creation. In: *Proc. Int. Conf. Entertainment Computing*, pp. 151–156. Springer, 2004. doi:10.1007/978-3-540-28643-1_19.

[39] A. Santamaría-Ibirika, X. Cantero, S. Huerta, I. Santos, and P. G. Bringas. Procedural playable cave systems based on Voronoi diagram and Delaunay triangulation. In: *Proc. 2014 Int. Conf. Cyberworlds*, pp. 15–22. IEEE, 2014. doi:10.1109/CW.2014.11.

[40] R. C. Silva, N. Fachada, D. De Andrade, and N. Códices. Procedural generation of 3D maps with snappable meshes. *IEEE Access*, 10:43093–43111, 2022. doi:10.1109/ACCESS.2022.3168832.

[41] The Elder Scrolls V: Skyrim, game webpage. `https://elderscrolls.bethesda.net/pl/skyrim10`, Accessed: 18.12.2023.

[42] R. Smelik, K. Galka, K. J. De Kraker, F. Kuijper, and R. Bidarra. Semantic constraints for procedural generation of virtual worlds. In: *PCGames '11: Proc. 2nd Int. Workshop on Procedural Content Generation in Games*, pp. 1–4, 2011. Article No. 9. doi:10.1145/2000919.2000928.

[43] R. Smelik, T. Tutenel, K. J. De Kraker, and R. Bidarra. Integrating procedural generation and manual editing of virtual worlds. In: *Proc. 2010 Workshop on Procedural Content Generation in Games*, pp. 1–8, 2010. doi:10.1145/1814256.1814258.

[44] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes. A survey on procedural modelling for virtual worlds. *Computer Graphics Forum*, 33(6):31–50, 2014. doi:10.1111/cgf.12276.

[45] R. M. Smelik, T. Tutenel, K. J. De Kraker, and R. Bidarra. Declarative terrain modeling for military training games. *International journal of computer games technology*, 2010, 2010. doi:10.1155/2010/360458.

[46] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra. A declarative approach to procedural modeling of virtual worlds. *Computers & Graphics*, 35(2):352–363, 2011. doi:10.1016/j.cag.2010.11.011.

[47] R. M. Smelik, T. Tutenel, K. J. de Kraker, R. Bidarra, et al. A proposal for a procedural terrain modelling framework. In: *EGVE (Posters)*, 2008.

[48] Speedtree documentation. `https://docs9.speedtree.com/`, Accessed: 22.04.2023.

[49] Home of Unity Technologies project. `https://unity.com/`, Accessed: 22.04.2023.

[50] Unreal Engine webpage. `https://www.unrealengine.com/en-US/`, Accessed: 22.04.2023.

[51] Unreal Engine 5.2 Roadmap. `https://portal.productboard.com/epicgames/1-unreal-engine-public-roadmap/tabs/85-unreal-engine-5-2`, Accessed: 22.04.2023.

[52] V. Valtchanov and J. A. Brown. Evolving dungeon crawler levels with relative placement. In: *C3S2E '12: Proc. Fifth International C* Conf. Computer Science and Software Engineering*, pp. 27–35. Montreal, Quebec, Canada, 27-29 Jun 2012. doi:10.1145/2347583.2347587.

[53] B. M. F. Viana and S. R. dos Santos. Procedural dungeon generation: A survey. *Journal on Interactive Systems*, 12(1):83–101, 2021. doi:10.5753/jis.2021.999.

[54] The Witcher game series webpage. `https://www.thewitcher.com/pl/en/`, Accessed: 18.12.2023.

[55] H. Yin and C. Zheng. A parametrically controlled terrain generation method. In: *Proc. 2012 7th Int. Conf. Computer Science & Education (ICCSE)*, pp. 775–778. IEEE, 2012. doi:10.1109/ICCSE.2012.6295187.

[56] Y. Zhang, G. Zhang, and X. Huang. A survey of procedural content generation for games. In: *Proc. 2022 Int. Conf. Culture-Oriented Science and Technology (CoST)*, pp. 186–190. IEEE, 2022. doi:10.1109/CoST57098.2022.00046.

Izabella Antoniuk received her MS from the Faculty of Applied Informatics and Mathematics at the Warsaw University of Life Sciences in the field of Computer Science, specializing in Multimedia Applications. In the same year began doctoral studies at the Faculty of Electronics and Information Technology of the Warsaw University of Technology. She obtained PhD in the field of: Technical Sciences and in the discipline: Computer Science in December 2018, with thesis entitled "Procedural generation of an editable three-dimensional terrain model based on selected features and schematic two-dimensional maps". In addition to working at the Warsaw University of Life Sciences she cooperates with other universities, as well as with business, implementing the EU applications. Her research interests include procedural content generation and advanced algorithms for use in computer games, artificial intelligence, machine learning, and data analysis.