CHALLENGES OF MODERN TECHNOLOGY | Vol. 7, No. 3, 2016, 8-15 DOI: 10.5604/01.3001.0009.5443

Procedural Generation of Underground Systems with Terrain Features using Schematic Maps and L-systems

Izabella Antoniuk, Przemysław Rokita
Warsaw University of Technology, Institute of Computer Science
Nowowiejska 15/19, 00-665, Warsaw, Poland
e-mail: I.Antoniuk@stud.elka.pw.edu.pl, P.Rokita@ii.pw.edu.pl

This article describes an algorithm for procedural generation of underground systems with terrain features, by processing set of schematic maps. L-system and cellular automata are used to generate final system shapes. Most of existing algorithms are not suitable for application in computer games, since they usually lack any considerable level of control, require large amount of computation or produce overly complex meshes. We present our solution, that can produce editable 3D objects from very simple input, with high level of control over final system layout. We also allow evaluation at every key step of generation process. Presented approach incorporates placement of various terrain features with stalactites, stalagmites and columns as an example of such elements. Obtained results can be used in computer games or similar applications.
Keywords: computer games, L-systems, procedural content generation.

Introduction

Procedural content generation algorithms provide user with diverse and visually acceptable content. Among such procedures especially those, that consider computer games as their main focus require specific constraints. Since any object intended for computer game needs to be well defined and generated according to 3D designer specifications, most of procedural algorithms are discarded in such applications.

Terrain generation for computer games is especially interesting and contains series of different problems. One of them is generating any type of terrain that cannot be represented by height map, such as caves [2]. Caves and dungeons are integral part of numerous computer games, starting with classic dungeon crawlers like Legend of Grimrock or Dungeon Master, to more recent productions like The Elder Scrolls: Skyrim or Witcher and Dragon Age Series. Elements like that require different constraints and additional rules, making them more difficult to model accurately, without losing control over final layout. Existing solutions usually only focus on creating 2D maps of such systems [20, 21]. 3D solutions usually require large amount of input data, long computation or do not provide designer with control over general shape of final system, while at the same time, obtained results are not easy to modify [1, 23-27].

In our approach we propose a method for generation of complex underground systems, that is based on set of schematic, 2D maps. By processing those files, we divide our system into tiles containing basic structures, such as caves, corridors or large spaces, connect them according to user specifications, to form larger network, and distribute simple terrain features. At the same time data representation we use permits overlapping structures. To ensure that generated terrain looks naturally, we use L-system to create basic shapes in each tile, extending it with cellular automata and ensuring accessibility of each part. Our approach can produce playable terrain from very simple input, with well-defined layout. We also permit modifications at every key step of generation process, as well as after generation.

The rest of this paper is organized as follows. Section 2 presents an overview of existing works related to our area of research. In section 3 we discuss initial assumptions of our algorithm. We present overview of our method in section 4. Obtained results and some areas of future work are presented in section 5. Finally we conclude our work in section 6.

Related work

While considering procedural content generation in the context of computer games and similar applications, we encounter different challenges and constraints, regarding final properties of desired output. Because of that existing algorithms can vary, both in type of objects they consider, and input required for generation algorithm. Depending from given requirements, such procedures can produce single type of objects (like roads [5], rivers [6] and vegetation [28]), chosen type of terrain [7, 8, 9] or entire worlds with different properties and constraints defining their shape [10, 11, 12].

Most of existing solutions provides user with content that is interesting visually. Main problem with such procedures is amount of control over final object shape, especially when considering elements related to game map layout. To apply such algorithms to computer games and similar applications, 3D designer should be able to define overall shape of final terrain, as well as its properties. Existing solution vary, from algorithms with only basic level of control [13], to procedures ensuring that final object will meet series of different properties [14]. There are also different ways in which such control can be provided, like using parameters to define shape of final object [9, 12], guiding generation process with story that will take place in created game world [16], providing simplified

shapes as input [7, 15] or using schematic maps, assigning each area of final terrain with different properties [17, 18].

Generation of various underground systems is another area of research related to creation of terrains intended for usage in computer games and similar applications. Caves are naturally occurring structures, that usually are formed when acidic water dissolves rock, and the resulting material is then removed by water flow [2]. By definition, caves are structures large enough for person to enter. They can also contain passages and spaces with characteristic elements that are often placed directly above one another. Because of that, resulting terrain data cannot be stored in height map. Most procedural algorithm would also require additional changes, before they can be applied to generation of such elements. There are few main groups of procedures, that are applied to generation of underground systems and related elements.

First group of algorithms focuses only on generation of various structures, characteristic to such systems, like overhangs [8], stalactites, stalagmites and columns [19]. Such algorithms can be incorporated in larger solutions and used to further increase realism of final terrain, or to add some details, interesting from gameplay point of view.

Second group considers generation of underground system in two-dimensional space. Different methods are used to define shape of created system, such as:

- Cellular automata applied to shape generation [20].
- Fitness function used to organize predefined shapes into entire system layout [21] .
- Fitness function that generated content with user-defined check points as its guide [22].

Although those algorithms produce interesting content, main problem with their application is that presented solutions are two-dimensional. If generated maps were converted to 3D terrain, they might present user with repeatable structures, and can also create areas that are not connected to main system. Also, neither of those approaches allows designer to define general layout of resulting system.

Final group of algorithms considers generation of underground systems in three-dimensional space. Existing solutions usually take some properties of final object as their main focus, such as:

- Geological correctness [24, 25]
- Realistic features [23]
- Ease of definition for final cave shape [27]
- Playability and level of control over terrain properties [26]
- Amplitude and playability of final systems [1].

Algorithms in final group can produce content that is diverse and interesting. Unfortunately they still have some drawbacks. Although results obtained in [24, 25] are geologically correct and interesting visually, they also are difficult to modify and incorporating them in computer game or similar application might be hard. Algorithm presented in [23] can produce very interesting content, especially from gameplay point of view. Unfortunately final mesh of object is very complex. Also, voxel representation used by authors in generation process leaves some remnants in object shapes, despite smoothing procedure that is applied to blur such elements. Method presented in [27] provides user with very high level of control over underground system shape. Authors use simple shape to design cave shape and then generate it accordingly. At the same time, creating more complex systems would require large amount of work. This approach also doesn't consider any terrain properties that are important for computer games and similar applications. Such properties are taken into account in [26]. Authors make sure, that generated system is playable. User can also define most of important properties of final terrain, such as size of generated spaces, relations between different structures or number of branches that will be modelled. Unfortunately, due to voxel representation used to define space, incorporating this solution in most computer game-like applications would require additional work. Also, despite fact, that user can define different system properties, its layout still is mostly random, without any significant way to control it. Authors also point out, that generation time is sometimes difficult to predict, especially with more complex systems. Second approach that also considers playability of generated terrain, uses slightly different approach [1]. Authors first use L-system to define structural points. In second steps, tunnels and caves are generated by wrapping meta-ball around defined paths. Finally, mesh data of underground system is obtained from voxel representation and everything is rendered with applied textures and shaders. Generated systems are complex and vast, unfortunately since authors cover large areas with same texture, they can appear repeatable. Authors also provide no way to define overall layout of final system, apart from using data obtained from L-system.

Procedural generation provides ways to generate different terrains, with various elements and properties. For detailed study of existing algorithms see [3, 4, 28, 29, 30, 31].

Initial Assumptions

Algorithms used in procedural content generation usually vary greatly, regarding type and amount of input data or manner in which information is processed. In our procedure we assumed that designer should be able to incorporate final object in computer game or similar application. Taking that into account we wanted to achieve high level of control over final system shape. Another important issue was controlling every key step of generation process therefore avoiding propagation of different errors. We also wanted to provide designer with easy way to modify final terrain.

In our previous work [32] we generated terrain using set of schematic maps. Since except for a few specific situations, terrain in computer games is usually represented by 2D map, that player can refer to (i.e. underground maps in Dragon Age: Inquisition, where terrain was represented by layered maps, outlining cave shapes), we decided to use similar approach.

Our system is represented with set of schematic maps, where each pixel represents properties of single tile in final object. We use tree types of maps: terrain map, height map and connection map. Our system is divided into levels and tiles. Single level contains tiles with same height dispersion, without overlapping. Terrain map, represented as RGB image file, describes properties of terrain in processed tile. For example, depending from type of area we would chose different parameters for generation algorithms. Height map describes relative heights of terrain tiles across entire scene. It is stored in

grey-scale image. Connection maps, stored in text file, define transitions between terrain tiles. We use two of such maps, where first defines connections inside each level of terrain, while second outlines transitions between levels. Tiles are defined as squares, to better represent pixels from input maps and simplify calculations. Tile size defines number of vertices in 3D object, limiting amount of detail that can be generated. Each of input maps can be either provided by designer, or generated automatically. We also store data from all terrain levels in single file (see Figure 1).

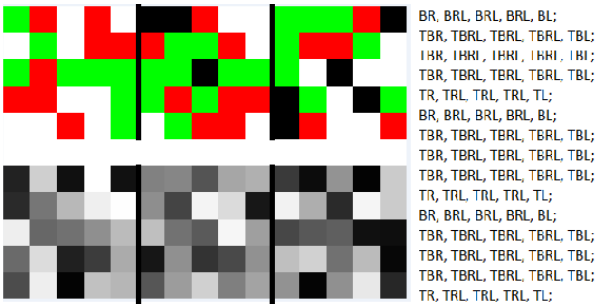

Figure 1. Input files used by our procedure. (Top left) Terrain map: each colour represents different terrain type. (Bottom left) Height map. (Right) All possible connections for given set of data (T – top, B – bottom, R – right, L – left). Vertical lines on terrain map and height map mark transitions between following level data.

Algorithm Overview

In this section we would like to present overview of our algorithm, along with used parameters and values. In second part we also describe separate procedure, that creates and places simple features (stalactites, stalagmites and columns) across generated cave system.

Terrain generation

Presented algorithm was implemented in Blender application (for Blender documentation see [33]). We decided to use this 3D modelling environment, because it contains complete python interpreter, providing easy access to program functionality. We can adjust it easily to visualize results of our procedure work, and since it is a modelling application, we can also check or change obtained objects. Currently our algorithm consists of two main steps:

- Processing input files and generating 2D tile maps.
- Generating 3D terrain.

Basic shape of terrain can be defined with different procedures, where some require simple input [13, 17, 18], while other are more complex [7, 10, 11, 12, 15]. In our approach we decided to use L-system for basic cave shape generation, since such algorithms tend to produce elements that look naturally, adding to final terrain believability. After initial shape is produced, we then apply cellular automata, to further fill and extend it. We use second algorithm because its properties can be easily adjusted to further outline desired terrain properties.

To assign properties of both algorithms we use data obtained from input files. Frist map we process is terrain map. Data stored in this image describes type of terrain in each tile. Single pixel corresponds directly to single region in final terrain, defining type of area and its properties. Depending from that information, parameters such as number of iterations, both for cellular automata and L-system, will differ. For our system generation we defined four types of tiles, assigned to different colours in input file:

- White: large space,
- Red: small space,
- Green: passage and
- Black: empty tile.

Highest iteration values are assigned to large spaces, while passages have lowest values of that parameter.

Expansion rules for our L-system are generated randomly for each given set of data, unless they are provided by user. Basic set of keys contains the following elements:

- T: Go to top cell
- B: Go to bottom cell
- R: Go to right cell
- L: Go to left cell
- I: Increase cell height
- D: Decrease cell height
- R: Reset cell height
- S: Create space

In first step of this part of our procedure, we generate set of tile maps for each level of final system. Tile size value, that is assigned by user, indicates number of pixels in each tile image (each pixel will refer to single vertex in 3D object later on). We create set of files containing tile maps and then fill them with evolved L-system. At this point we also check connections defined between different tiles, since starting points for this part of algorithm are set at tile edges, that are connected to neighbouring tiles according to both connection maps. We also place single L-system at the middle of the tile, to better fill it and produce more interesting shapes. At this point we can see basic shape of currently processed tile (see Figure 2.).

Figure 2. Evolution of single tile during generation process. Starting from left: generated L-system, tile processed by cellular automata, tile after connecting spaces and tile after final smoothing of cave shapes.

Terrain generated only by L-system presents inconsistent and ragged appearance. In next step we apply cellular automata algorithm to fill and extend spaces inside tile. At this point we usually had few nice looking spaces representing fragment of our cave system. We then connect those spaces, ensuring that each part of generated terrain will be accessible. Finally we apply smoothing procedure, widening to narrow passages and further filling cave shape. Example tile maps generated for single level with different tile sizes are presented at Figure 3. It can be noticed, that sometimes, especially when L-system circulates some local minimum close to edge, tile borders are visible. Despite that fact, generated systems are interesting visually, and often quite close in their general layout to real world caves (see [2]).

At this point, our approach has following advantages, when compared to other, 2D solutions:

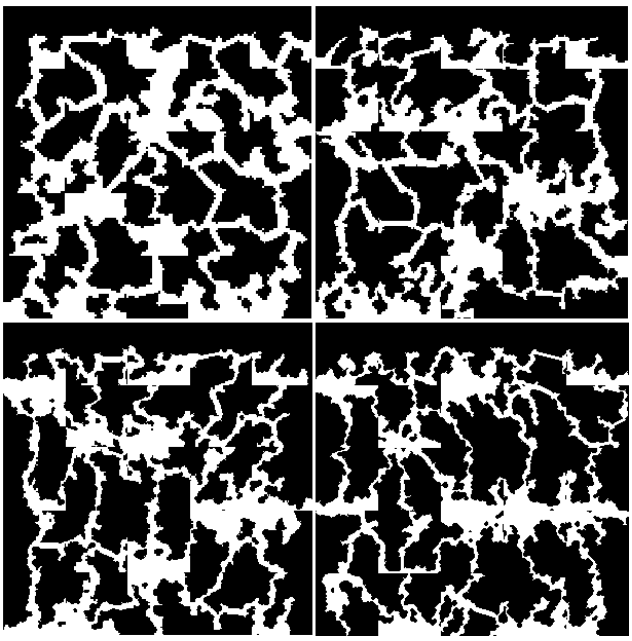

Figure 3. Example tile maps generated for single level, with different tile sizes: 31x31 (top left), 41x41 (top right), 51x51 (bottom left) and 61x61(bottom right).

- We connect all spaces in our system, avoiding generation of isolated areas that cannot be accessed from main system.
- Since we are not using predefined shapes, we are not limiting possible outcomes, and generated layouts are more interesting.
- Schematic maps we use as an input for our procedure give an opportunity to define overall layout of final system.
- Our data representation is fully three-dimensional, with easy way to define transitions and relations inside cave system.
- Because of data generated by L-system, 3D objects obtained from our maps are not repeatable.

After generating tile maps for all levels, we use them, along with calculated cell-heights and information from input files to model 3D representation of underground system. Since Blender application usually operates on square faces, not triangular ones, we transform our tile maps into grids with given size. Number of pixels in each image corresponds directly to amount of vertices in its 3D representation. Each tile from terrain map will be represented as grid object (unless it was defined as empty) and placed in scene according to data obtained from height map. We use this operation to show general layout of our system and height transitions between tiles. When grids are placed and general layout accepted by user, we proceed to assigning cave shapes and heights generated for each tile to those elements. We use values from tile maps generated in previous step to achieve that. To better visualize our underground system, we remove vertices that represent wall cells. Since resulting object presents jagged appearance, we fill any sharp transitions that remained after removing wall vertices. Each vertex that represents cells classified as cave is then placed at appropriate height, to show cave shape.

At this point of generation, we have set of tiles with assigned cave shapes, but there are no connections between those tiles yet. Using the same principle as in our terrain generation algorithm (see [32]), we start by connecting edges of those tiles, that have defined connection between them. At the same time we perform smoothing procedure, that starts its work at the edge of a tile, and proceeds deeper into considered tiles. Since L-system and cellular automata sometimes produce heights that are very scattered, at this point we smooth those transitions, by using simple 2D filter. Finally we apply simple material and texture to generated object (see Algorithm 1).

Algorithm 1. Procedural underground generation

```
procedure CaveGeneration(TileSize, Size, Height-
Map, ConnectionsTiles, ConnectionsLevels, Ter-
rainMap, NumberOfLevels)
  for all Tiles in TerrainMap do:
    UpdateTileWithL-system(TileMap)
    UpdateTileWithCellularAutomata(TileMap)
    SmoothMaps(TileMap)
  end for
    PlaceBasicGrids(HeightMap)
  for all Grids do:
    UpdateGridData(TileMap, VertexHeights)
    AllignAndSmoothBorders(TileHeights, TileMaps,
    TileConnections, Depth)
    AssignMaterial(name)
    AssignTexture(name)
  end for
  for all Regions do:
    GenerateFeatureLocations(number)
    GenerateAndPlaceFeatures(locations)
  end for
end procedure
```

Generation and placement of terrain features

One of most common feature, that can be encountered in cave-like underground systems are stalactites, stalagmites and columns. They can come in many different shapes, and can be placed both in passages, and larger spaces, creating obstacles and adding to underground system appearance.

First operation we perform during this part of procedure, is finding locations where we will place our terrain features. We start by randomly choosing some of vertices as bases for our calculations. We then calculate actual location along cave edge.

Once we have set of locations, we then proceed to assigning type of element that will be placed there (stalactites, stalagmites or columns). We can either limit number of single type of feature, or ensure, that it will reach certain value.

Finally, for each defined feature, we generate object, that will represent it. Apart from location and type, each element is also represented by values describing tip and base diameter, as well as overall height and number of vertices forming base. With such approach we can generate vast collection of stalactites, stalagmites and columns, where the last ones are generated as combination of the first two.

With all those values set, we start by checking type of generated element and placing base circle (or circles in case of columns) at appropriate places. Each of vertices is then displaced, to distort shape of generated element. Overall height of this element describes two values: size of speleothems along z axis and number of lines that will form its shape. We place following circles from base to tip, gradually reducing diameter according to given limits. Finally we fill both ends of created

speleothems and assign simple material and texture. Example cave features generated with our algorithm see Figure 4. Maximum height is set at 12 lines, and maximum diameter in vertices is set at 32.

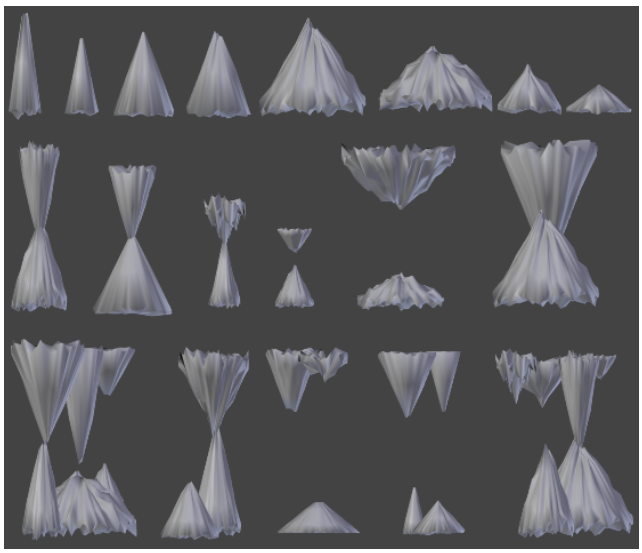

Figure 4. Example structures generated by our speleothems generation procedure. Top line: examples for different height and base/tip diameter. Middle and bottom line: different layouts obtained by combining basic shapes.

At this point we have full underground system, along with different elements placed. Due to form of our algorithm, we can easily replace speleothems used in that example with any other object, either generated or provided by user.

Although our algorithm doesn't present user with results at interactive rate, its computation time is still acceptable and mostly constant for given size of data set (see Table 1). Since both terrain and inserted features are Blender mesh objects, they can be also easily modified. Each of those objects can also be regenerated, therefore one faulty part doesn't exclude entire result. Since our approach is seed based (we use it both in tile map generation and while placing features through scene), we can also acquire different variations of terrain for given set of data. Example terrains along with placed features are presented at Figure 5. Terrains were generated by using schematic maps containing single level data in case of first three objects, three levels in case of fourth object and two levels in case of last object.

Table 1. Rendering times (in seconds) for different tile sizes. I – L-system iterations, M – generation time for tile maps, 3D – generation time for 3D terrain, F – generation time for terrain features. System has 3 levels 5x5 tiles each.

Tile size	I	M	3D	F	Total time [s]
21x21	2	15.51	4.32	2.15	21.98
31x31	3	63.21	9.02	4.95	77.18
41x41	4	155.81	23.16	12.24	191.21
51x51	5	441.12	51.53	28.32	520.97
61x61	6	917.05	72.30	39.44	1028.79

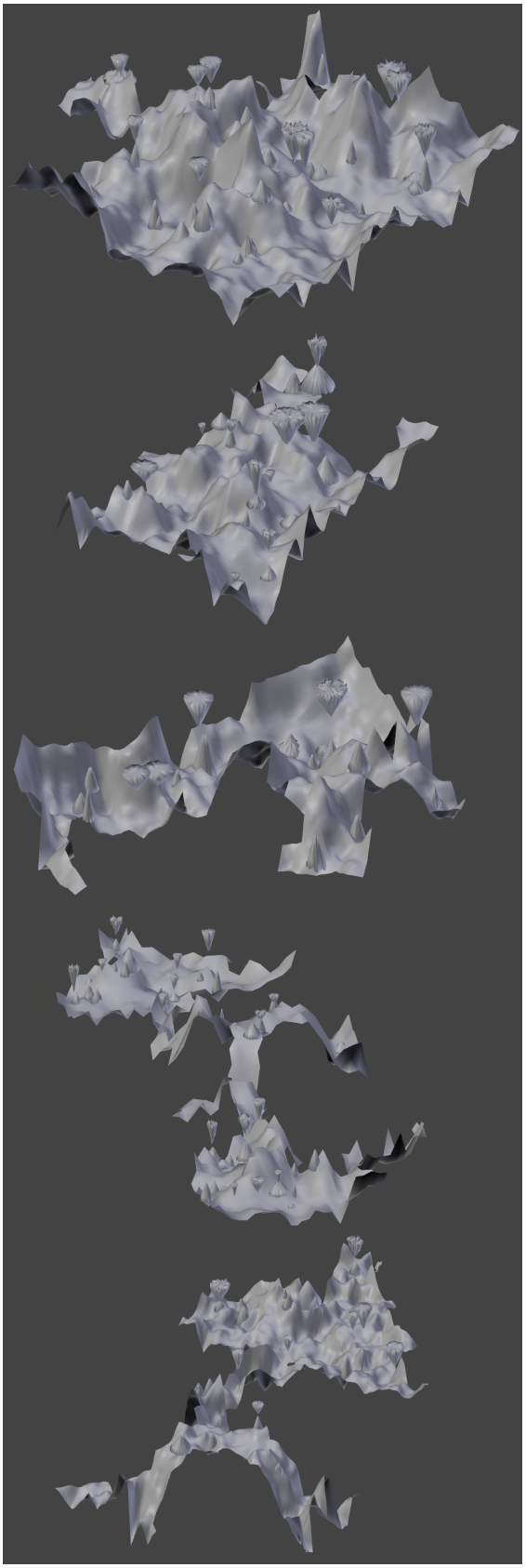

Figure 5. Example terrains generated by our procedure. Three top objects show single level caves while two bottom elements illustrate multilevel systems.

For overview of our procedure with visualization of all key steps see Figure 6.

Results and future work

We implemented algorithm presented in this paper using python interpreter incorporated in Blender. Resulting application can generate underground, cave-like systems and represent them as mesh objects. All experiments were performed on a PC with AMD Radeon HD 6650M graphic card, Intel Core i5-2430M processor (2.4GHz per core), and 4GB of ram.

During our experiments we tested all critical factors and how they affect our generation procedure. Few properties of those parameters became apparent during those experiments. First set of values concerns generation of tile maps. While testing different values we found the following dependences:

- Optimal value for maximum length of L-system transition rules were found between values equalling one-fourth and half of tile size. Higher values usually resulted in L-system filling entire tile, while lower values produced repeatable shapes (see Figure 7).
- Number of L-system iterations, that produced interesting results, without increasing computational costs to greatly was obtained by taking floor from dividing tile size by ten (see Figure 8). Smaller values usually resulted in cluster, that stuck to starting cell and its neighbourhood, while larger values tended to fill entire tile and greatly increased computational costs (see Table 2).
- Threshold for cellular automata worked best when set to half of total cell neighbours or higher. We used that procedure to add some volume to generated terrain, but wanted to avoid situations, when it blurs shape generated by L-system and this values achieved exactly that (see Figure 9).
- Maps with single level size set at 5x5 tiles were best when we considered amount of work required to complete input files by hand, compared to level of control that they provide.

Map generation times (in seconds) for different number of L-system iterations. Single tile size is set at 21 and scene is set at single level 3x3 tiles.

L-system iterations	Map generation time [s]
2	1.774
3	1.955
7	3.721
8	5.989
9	14.021

Second set of parameters concerned generation of 3D object and speleothems and placement of those features across the scene:

- To large height difference between regions in final object resulted in connections that were to sharp and looked unnaturally. To avoid this, difference between basic heights of terrain regions should be smaller than maximum region height.
- Number of vertices set for single circle in stalactite, stalagmite or column should be greater than 10, otherwise generated shapes are very simple and look unnaturally.

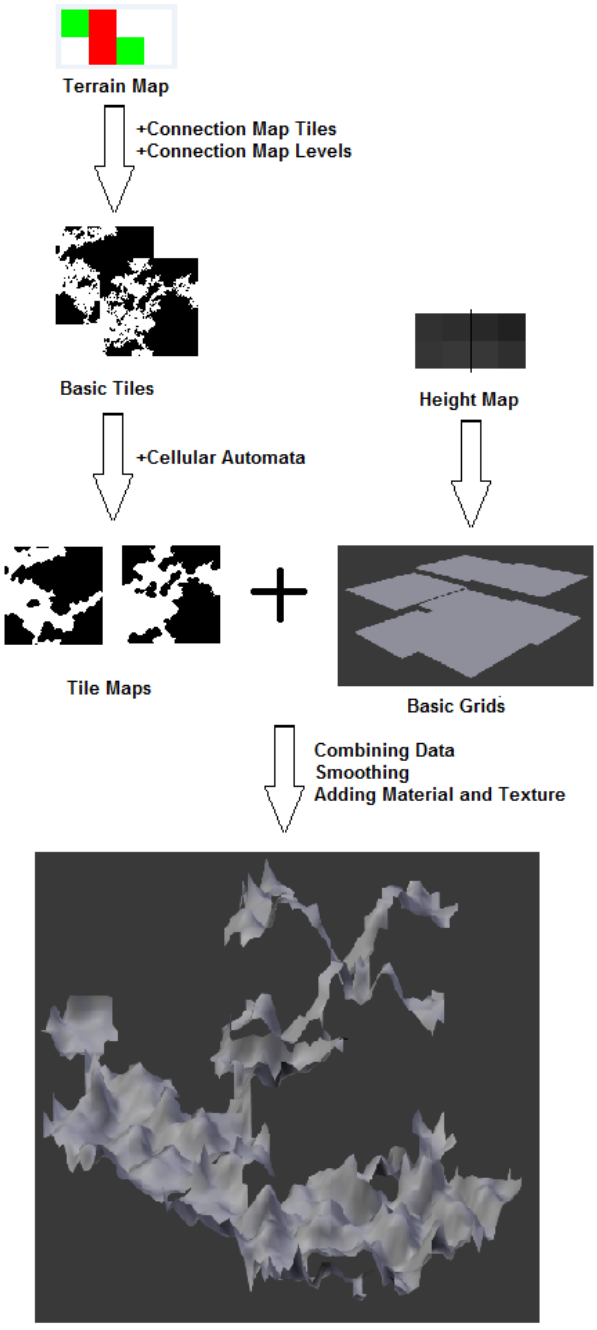

Figure 6. Algorithm overview.

Figure 7. Resulting L-systems with different maximum rule length.

- Depending from chosen set of data, our algorithm can produce vast array of speleothems. In general data sets with large relative height values compared to base di-

ameter produced thin and sharp elements, while opposite set of data produced short and flat objects.

Main application that we consider for terrains generated by presented algorithm is to incorporate them in computer game or similar application. They can be used either as a base for terrain in simple game, or provide good visualization during design process. Our approach allows for terrain generation with high level of control. Also created elements are interesting visually, and although sometimes tile structures are visible, such problems can be easily solved, either by regenerating selected tile with different seed or by adjusting it manually. In those instances, where generation rules are generated randomly, there are cases when created content is repeatable or sticks to one point. This problem can also be easily solved, by changing seed base for that element. Because entire system is divided into separate objects, it can be easily adjusted, either inside Blender, without any additional operations, or by converting output meshes to other 3D modelling environments. Finally our method for placing features can easily distribute different elements across underground system (in this case we place only speleothems), while procedure for stalactites, stalagmites and columns generation returns various and visually interesting objects.

Our approach still has some drawbacks. First of all, generated terrain still contains little details. We would like to improve generation procedure, as well as incorporate separate algorithm for generating textures instead of using one of inbuilt Blender functionalities in that aspect. Another issue we would like to solve is visibility of borders between tiles. To sharp transitions between regions in 3D object can be solved by adding separate procedure, that will generate such passages. As for our performance it can be further improved, for example by incorporating parallel computation.

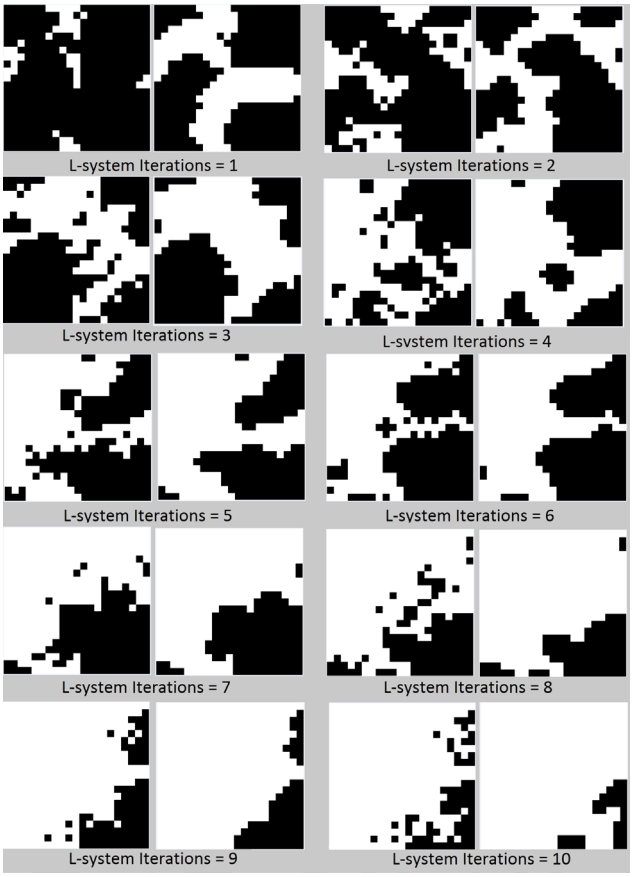

Figure 8. Resulting L-systems with different number of iterations and corresponding results after applying cellular automata.

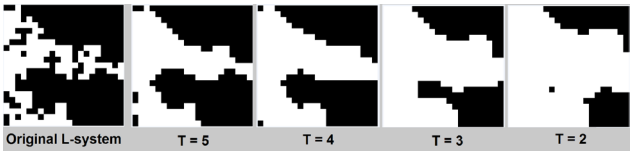

Figure 9. Resulting shapes with different cellular automata threshold. Original L-system is on the left.

Conclusions

In this paper we propose a method for procedural generation of underground systems with terrain features. We use schematic maps to provide data for L-system and cellular automata for generation of basic shape of terrain, and then apply this information to generate 3D object.

Contrary to existing solutions [23-27] we take into account constraints that are specific to computer games, maintaining connectivity of all spaces inside our system, and limiting final mesh complexity. By using schematic input maps we ensure, that final object will be as close to user specifications as possible, with easy way to define its overall layout, contrary to some other solutions [1, 26]. Maps we use are also designed to represent 3D terrain easily. Designer can also evaluate algorithms progress at every key step.

Main problems we encountered was too slow computation with greater number of L-system iterations, or occasional visibility of borders between terrain tiles. We plan to address this problems in future work. We would also like to add more terrain features, characteristic for computer games. Although our approach still has some drawbacks it can produce usable terrains that can be either incorporated in simple game, used as a base for further work or for visualization purposes during design process.

References

[1] Mark B., Berechet T., Mahlmann T., Togelius J.: Procedural Generation of 3D Caves for Games on GPU, In Foundations of Digital Games, 2015.

[2] Palmer A. N.: Origin and Morphology of Limestone caves, Geological Society of America Bulletin, 103(1), 1-21, 1991.

[3] Shaker N., Liapis A., Togelius J., Lopes R., Bidarra R.: Constructive generation methods for dungeons and levels (DRAFT), Procedural Content Generation in Games, 31-55, 2015.

[4] Van der Linden R., Lopes R., Bidarra R.: Procedural generation of dungeons, IEEE Transactions on Computational Intelligence and AI in Game, 6(1), 78-89, 2014.

[5] Galin E., Peytavie A., Marchal N., Gurin E.: Procedural generation of roads, Computer Graphics Forum (Vol.29, No.2, pp.429-438), Blackwell Publishing Ltd., 2010.

[6] Huijser R., Dobbe J., Bronsvoort W. F., Bidarra R.: Procedural natural systems for game level design, SBGAMES, pp. 189-198, IEEE, 2010.

[7] Kamal K. R., Kaykobad M.: Generation of mountain ranges by modifying a controlled terrain generation approach, 11th International Conference on Computer and Information Technology, pp. 527-532, IEEE, 2008.
[8] Gamito M. N., Musgrave F. K.: Procedural landscapes with overhangs, 10th Portuguese Computer Graphics Meeting, Vol. 2, p. 3, 2001.
[9] Michelon de Carli D., Pozzer C. T., Bevilacqua F., Schetinger V.: Procedural generation of 3D canyons, SIBGRAPI, pp. 103-110, IEEE, 2014.
[10] Peytavie A., Galin E., Grosjean J., Merillou S.: Arches: a framework for modeling complex terrains. Computer Graphics Forum, Vol. 28, No. 2, pp. 457-467, Blackwell Publishing Ltd., 2009.
[11] Smelik R. M., Tutenel T., de Kraker K. J., Bidarra R.: A declarative approach to procedural modeling of virtual worlds, Computers&Graphics, 35(2), pp.352-363, 2011.
[12] Smelik R., Galka K., de Kraker K. J., Kuijper F., Bidarra R.: Semantic constraints for procedural generation of virtual worlds, Proceedings of the 2nd International Workshop on Procedural Content Generation in Games, p. 9, ACM, 2011.
[13] Prusinkiewicz P., Hammel M.: A fractal model of mountains with rivers, Graphics Interface, Vol. 93, pp. 174-180, Canadian Information Processing Society, 1993.
[14] Tutenel T., Bidarra R., Smelik R. M., de Kraker K. J.: Rule-based layout solving and its application to procedural interior generation, CASA Workshop on 3D Advanced Media In Gaming And Simulation, 2009.
[15] Merrell P., Manocha D.: Model synthesis: a general procedural modeling algorithm. IEEE Transactions on Visualization and Computer Graphics, 17(6), p 715-728, 2011.
[16] Matthews E., Malloy B.: Procedural generation of story-driven maps CGAMES, pp. 107-112, IEEE, 2011.
[17] Smelik R. M., Tutenel T., de Kraker K. J., Bidarra R.: A proposal for a procedural terrain modelling framework, EGVE, pp. 39-42, 2008.
[18] Smelik R. M., Tutenel T., de Kraker K. J., Bidarra R.: Declarative terrain modeling for military training games, International journal of computer games technology, 2010.
[19] Raz Tortelli D. M., Walter M.: Modeling and Rendering the Growth of Speleothems in Real-time, GRAPP, pp. 27-35, 2009.
[20] Johnson L., Yannakakis G. N., Togelius J.: Cellular automata for real-time generation of infinite cave levels, Proceedings of the 2010 Workshop on Procedural Content Generation in Games, p. 10, ACM, 2010.
[21] Valtchanov V., Brown J. A.: Evolving dungeon crawler levels with relative placement, Proceedings of the 5th International C* Conference on Computer Science and Software Engineering, pp. 27-35, ACM, 2012.
[22] Ashlock D., Lee C., McGuinness C.: Search-based procedural generation of mazelike levels, IEEE Transactions on Computational Intelligence and AI in Games, 3(3), pp. 260-273, 2011.
[23] Cui J., Chow Y. W., Zhang M.: Procedural generation of 3D cave models with stalactites and stalagmites, 2011.
[24] Boggus M., Crawfis R.: Explict Generation of 3D Models of Solution Caves for Virtual Envinronments, CGVR, pp. 85-90, 2009.
[25] Boggus M., Crawfis R.: Procedural creation of 3d solution cave models, Proceedings of IASTED, pp. 180-186, 2009.
[26] Santamaria-Ibirika A., Cantero X., Huerta S., Santos I., Bringas P. G.: Procedural Playable Cave Systems based on Voronoi Diagram and Delaunay Triangulation, International Conference on Cyberworlds, pp. 15-22, IEEE, 2014
[27] Boggus M., Crawfis R., Prismfield: a framework for interactive modeling of three dimensional caves, Advances in Visual Computing, pp. 213-221, Springer Berlin Heidelberg, 2010.
[28] Prusinkiewicz P., Lindenmayer A.: The algorithmic beauty of plants, Springer Science & Business Media, 2012.
[29] Hendrikx M., Meijer S., Van Der Velden J., Iosup A.: Procedural content generation for games: A survey, ACM TOMM, 9(1), 1, 2013.
[30] Smelik R. M., Tutenel T., Bidarra R., Benes B.: A survey on procedural modeling for virtual worlds, Computer Graphics Forum, Vol. 33, No. 6, pp. 31-50, 2014.
[31] Ebert D. S.: Texturing & modeling: a procedural approach, Morgan Kaufmann, 2003.
[32] Antoniuk I., Rokita P.: Procedural Generation of Adjustable Terrain for Application in Computer Games Using 2D Maps, Pattern Recognition and Machine Intelligence, pp 75-84, Springer International Publishing, 2015.
[33] Blender application home page: https://www.blender.org/ (Accesed 12.06.2016).