Procedural Generation of Virtual Cities

Adam Brol
Institute of Information Technology
Warsaw University of Life Sciences
Warsaw, Poland
adam.brol10@gmail.com

Izabella Antoniuk
Institute of Information Technology
Warsaw University of Life Sciences
Warsaw, Poland
izabella_antoniuk@sggw.edu.pl

***Abstract*—In this article a comprehensive methodology for procedural generation of cities is introduced. The method consists of three distinctive stages for terrain, road network and building generation. Presented algorithm is flexible and customizable, providing user with possibility to adjust the generation process and final outcome to their specific needs. Two methods were implemented for each stage and later utilized to generate a collection of virtual cities, demonstrating the versatility of proposed approach.**

***Index Terms*—procedural generation, procedural modelling, terrain generation, city generation, virtual cities**

I. Introduction

Modern film productions, computer games and other three-dimensional visualizations often require a believable and aesthetically pleasing model of a city with dozens of streets and hundreds of buildings. The traditional approach of manually crafting these cities by artists becomes increasingly impractical as the scale of the project expands. Consequently, there arises a need for automated methods that can streamline this process.

Procedural content generation describes solutions for algorithmic data creation, using a set of rules, with limited human input. Depending on the specific application and implementation, procedural generation offers several advantages:

- Smaller file size: Procedural generation algorithms are significantly smaller in size compared to the resulting data, reducing disk space used.
- More content: By leveraging the random nature of procedural generation, it is possible to generate an almost unlimited number of different objects, sharing similar properties but visually different from each other.
- Variety: Particularly relevant to computer games, when large quantities of similar objects need to be created, set generated by algorithms will usually be more varied than the human-created one.
- Short iteration time: Modifying parameters or rules allows for easy modification of the results.

The main challenge of creating virtual cities, is their complexity and scale. The manual modeling of terrains, intricate street networks, and hundreds of buildings is an incredibly laborious task. Furthermore, making changes to a completed model requires manual adjustments. Major modifications to the original specifications can easily lead to models being discarded and entire work starting from scratch. These issues can be solved with procedural generation techniques.

II. Procedural modelling

Procedural modelling is a very alluring field of study, offering the prospect of generating a vast array of detailed models in a matter of seconds. Over the past four decades, this area has been the subject of active research, resulting in numerous publications focusing on modeling specific features or structures. However, few have focused on complex systems, which necessitate the integration of multiple methods for different elements, such as cities.

A. Terrain generation

Terrain generation is one of the most common applications of procedural generation algorithms. Almost every three-dimensional computer game has terrain for the player to navigate. Open-world games require landscapes spanning tens of square kilometers, flight simulators demand thousands, and some games enable players to explore entire planets.

Popular terrain representation involves using heighmap. Most aspects of the terrain can be represented using two-dimensional matrices. Width and height correspond to the dimensions of the surface, and the values indicate the height of the terrain at a given point. This type of matrix is commonly referred to as a heightmap. It is important to note, that heightmaps possess a notable limitation. They are not capable of representing structures such as caves or overhanging sections of cliffs. This limitation arises from the fact that each point in the heightmap can only record a single height value.

A natural fit for generating heightmaps is noise. Numerous noise-based techniques have been employed in terrain generation, including simplex noise and the diamond-square algorithm. However, the prevailing method is the utilization of layered Perlin noise [1]. This approach has gained significant prominence, with notable implementation in the world generator of the immensely popular video game, *Minecraft*, where Perlin noise is used at multiple stages of the world generation process.

B. Road network generation

Road network generation plays a key role in the urban modeling process. It gives the city a distinctive appearance, distinguishing modern metropolises where roads form a regular grid, from historic ones where the roads were developed over hundreds of years, organically arranging themselves into a variety of shapes.

The simplest form of street layout is the regular grid system, commonly observed in modern cities. It's comprised of a network of roads intersecting at right angles. Even though the implementation of a generator for such a grid tends to be trivial, it can still finds interesting uses [2]. Different method, slightly more complex, involves placing roads along the edges of a Voronoi diagram. This approach produces a much more dynamic looking road network, though not a very realistic one. However, combining Voronoi diagrams with templates has proven to yield more satisfactory outcomes [3], [4].

In their work on the *CityEngine* city generation system, Parish and Müller employed an L-system to generate road networks [5]. The authors achieved impressive results, reproducing a variety of realistic street patterns. However, the use of an L-system introduces complexities in implementation that could be alleviated by adopting a simpler algorithm.

C. Buildings modelling

Once the road network is established, the final component required to complete a virtual city is the generation of buildings. Buildings are an integral part of the urban landscape and have a key impact on the character and appearance of the city. Procedural methods for building generation enable efficient creation of a diverse array of structures, while considering the road network and the shape of designated plots of land.

With an array of defined plot shapes, the simplest approach for generating three-dimensional buildings involves extruding the shape upward to the desired building height. Consequently, each generated building takes the form of a right prism, fitting the plot's shape. The main advantages of this method are its ease of implementation and computational speed. Major disadvantage is the limited variability of the resulting models. This method is more than sufficient for modeling cities consisting primarily of simple skyscrapers or apartment blocks.

A more advanced technique, as demonstrated by Greuter et al. [6], is the method of combining primitives. This process starts off with the creation of the topmost section, where the floor plan is formed by combining several primitives. Subsequent sections incorporate additional primitive shapes onto the previous floor plan. The body of of the building is formed by extruding each section downwards.

Parish and Müller employed L-systems not only for generating street networks but also for the building geometry [7]. In this approach, the L-system's axiom is represented by a prism. The alphabet consists of operations such as scaling, sliding, extrusion, while templates contain pre-designed elements like roofs or antennas. In their subsequent work, authors further improved on this approach with CGA shape, moving away from the text-based L-systems to directly manipulating geometric shapes. This shift allowed for more precise rulesets, offering greater control and resulting in more detailed buildings [9].

D. City generation

City generation is a complex problem that necessitates the integration of various procedural methods. At minimum, it can be divided into two stages: road network generation and building generation. Additionally, for three-dimensional applications, it is advisable to include a terrain generation stage.

Parish and Müller developed a city generation system that utilizes L-systems for both road network and building generation [7]. The city layout is derived from geographical and socio-statistical maps, rather than being explicitly controlled by the user. Their approach proves effective in creating realistic replicas of existing cities, but it may not be as suitable for constructing custom areas from scratch.

Smelik et al. [8] developed a world modelling system called *SketchaWorld*, which integrates user input with procedural methods. Users provide a sketch of the desired world, which the system uses to generate the requested features while ensuring they fit cohesively with the surrounding environment. Although this system relies heavily on procedural generation, it is considered a procedurally assisted modelling tool rather than a complete procedural generation system.

III. A UNIFIED MODEL FOR CITY GENERATION

As previously mentioned, few publications have developed systems for generating entire cities from scratch. Furthermore, the existing solutions restrict their users to specific generation methods implemented in closed source software packages. Replacing any of the generation methods with alternatives is not possible.

To address these limitations, we propose a comprehensive approach for the procedural generation of cities. Each step of the process is clearly defined by its input and output, promoting flexibility and customization. The pipeline consists of four distinct generation steps: terrain, roadmap, allotments, and buildings, as depicted in Figure 1. Each step is associated with a specific data type and format, facilitating the integration of different generation methods.

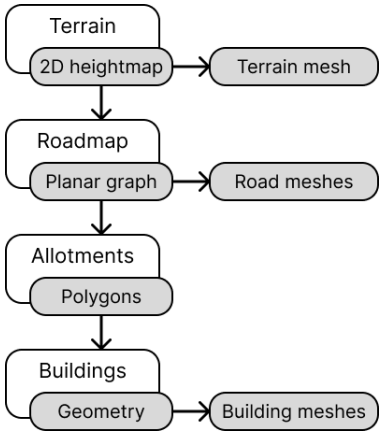

Fig. 1. Generation steps of the city generation approach.

To demonstrate the flexibility of the presented approach, two methods were implemented for each step. The results section presents a curated collection of cities generated using different combinations of these methods, demonstrating the diverse possibilities and potential outcomes.

A. Terrain heightmap

The use of heightmaps is a prevalent method for representing terrain data. While heightmaps have limitations in representing overhangs and caves, these features are typically irrelevant in the context of urban environments. Therefore, heightmaps remain a suitable choice for generating terrain for cities.

Regarding the dimensions of the heightmap matrix, it is important to adjust them according to the desired level of detail. This allows for flexibility in determining the resolution and granularity of the generated terrain, accommodating specific requirements and computational resources.

The heightmap values can be populated using the chosen method to create the desired terrain features. It is advisible, to keep the values within the normalized range, as this facilitates easier fine-tuning of the exact terrain heights.

Once the heightmap is filled with appropriate values, generating a mesh representation of the terrain can be achieved by utilizing a subdivided 3D plane. By offsetting the vertices of the plane to the corresponding altitudes calculated from the heightmap, a 3D model of the generated terrain is created.

B. Roadmap graph

The roadmap step involves the generation of a planar graph that represents the road network of the city. In this graph, nodes correspond to intersections and turns, while edges symbolize the roads connecting them. The planar nature of the graph doesn't limit the possibility of including structures such as overpasses in the road network, as these can be represented as a special kind of intersection.

The previously created heightmap can be used to determine how roads should be placed, for example to avoid placing roads at steep angles or at too high or low altitudes.

Once the road graph is generated with a selected method, the road meshes can be generated. By sampling the heightmap or the terrain mesh, the road meshes can be adjusted to conform to the terrain features.

C. Allotments and buildings

Before buildings can be placed in the city, we first need to determine which areas are valid for construction. We define these spaces as allotments, sections of the city that are surrounded by roads from all sides. These allotments are represented as polygon shapes, which are created by insetting closed sections of the road graph. The inset amount is equal to half of the road width.

When generating the geometry for buildings, the edges of the allotments serve as boundaries to avoid intersection with road meshes. The specific placement of buildings within the allotments may vary depending on the chosen generation technique.

IV. Implementation and results

In order to validate the effectiveness of the proposed pipeline, a complete city generator has been developed using Unreal Engine 5. We utilized C++ to implement two distinct methods for each of the terrain, roadmap, and buildings generation stages.

A. Terrain generation

For the terrain generation stage, firstly a method based on Perlin noise was implemented. Unreal Engine provides a convenient function for sampling Perlin noise, however, to achieve visually pleasing terrain, it was necessary to combine multiple layers of noise with varying frequencies. The number of layers, the noise scale and the frequency and amplitude change ratio can each be adjusted.

Another popular technique for procedural terrain generation is the diamond-square algorithm. The process begins by initializing the values of the matrix corners. To fill the entire matrix, iterative diamond and square steps are performed.

During the diamond step, the algorithm finds the midpoint of the current square (initially encompassing the entire terrain) and sets its value as the average of the four corner values, plus a random value. The square step applies to the four midpoints of the current square and sets their values to the average of their two neighboring corners, the value of the center point already calculated in the diamond step, plus a random value.

After these steps, the terrain is divided into four equal parts, and the diamond and square steps are repeated for each part. This division continues until the entire matrix is filled.

After each terrain subdivision, the range of random values is scaled by a predefined roughness parameter, ranging from 0 to 1. This parameter controls the level of terrain detail, where higher values result in increased irregularity, while lower values produce smoother terrain.

B. Road network generation

The first technique chosen for road network generation was based on Voronoi diagrams. Unreal Engine provides the Voronoi.h header file that includes all the necessary methods for generating Voronoi diagrams. To generate the diagram, a set of control points is needed, with each point representing a cell in the diagram. These can be randomly placed on the terrain to create a natural cell-like structure, or uniformly distributed to form a grid layout.

The second, implemented method was a graph growing algorithm. Starting from a single point, the road network stretches out and spawns off new branches. The separate branches eventually connect with each other, resulting in a connected road graph.

Once the road graph is generated using either method, the spline tools provided by Unreal Engine is utilized in order to lay out the road meshes along the edges of the graph. To ensure that the roads match the environment, the terrain mesh is sampled and the street meshes is aligned to conform to its shape.

C. Building generation

For the building generation stage, firstly the extrusion approach is used. This method involves taking the shape of the allotment and extruding it vertically to create a building in the

form of a right prism. By applying appropriate textures and materials, visually convincing buildings can be generated.

In addition to the extrusion method, the shrinking layers technique for building generation was also implemented. This method builds upon the extrusion approach by adding multiple layers on top of the initially extruded shape. With each layer, the floor plan is progressively shrunk, resulting in more complex building structures.

D. City generation

Once the methods for all steps have been implemented, integrating them into a city generation system becomes straightforward. The modular design of presented methodology enables easy replacement of any component used in the generation process with an alternative method if desired. Algorithm 1 shows the overview of the generation with used input data.

```
func generateCity():
  heightmap = generateHeightmap()
  terrain = generateTerrain(heightmap)
  roadGraph = generateRoadGraph(heightmap)
  roads = generateRoads(roadGraph, terrain)
  allotments = generateAllotments(roadGraph)
  buildings = generateBuildings(allotments)
```

Algorithm 1. City generation pipeline.

Using the implemented methods, a diverse collection of virtual cities can be created. In Fig. 2, a city generated using the diamond-square algorithm for the terrain, a Voronoi-based road network, and 150 buildings generated with the shrinking layers method is shown. Fig. 3, presents another city with a flat terrain, a grid-based road network, and 3600 buildings generated through the extrusion method. The generation process in both cases took under 30 seconds (from initial setup, to the visual representation showing in the game engine). Presented approach can create diverse city layouts with distinctive features.

V. Conclusions

In this paper a comprehensive, modular approach for procedural city generation was presented. The process is divided into distinctive stages. Each of those stages can be achieved by using different algorithms, with the following stage incorporating previously obtained results. Two methods were implemented for each step, with the possibility to add different algorithms for content generation.

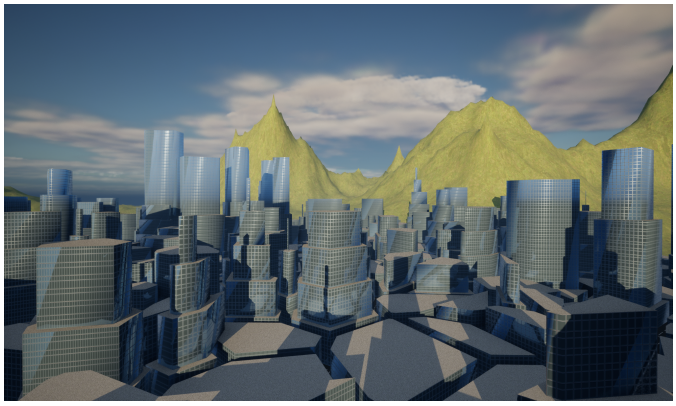

Fig. 2. Virtual city generated using the following methods: diamond-square for the terrain, a Voronoi diagram for the roadmap and the layering method to generate 150 buildings.

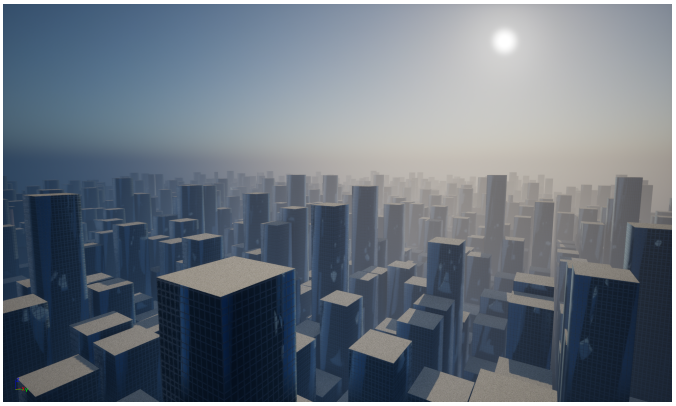

Fig. 3. Virtual city generated using the following methods: flat terrain, a Voronoi diagram for the roadmap and the extrusion method to generate 3600 buildings.

The presented methodology, if needed, can be expanded with additional stages. Possible areas of future research can include adding city zoning that would assign specific zones to allotments such as residential, commercial, or industrial, with different generation parameters. The zones could dictate the the kinds of buildings that can be generated in different areas of the city. Another potential extension is the addition of water bodies and vegetation, placing rivers, lakes and forests or parks around the city.

Overall, the results show, that the approach is flexible, easy to expand, and even in its initial form can produce interesting, diverse and visually appealing results. User has full control over each of the steps and can replace any of the implemented methods at will. Generated cities are ready to use and easy to modify due to their implementation in the Unreal Engine 5.

References

[1] G. Shaker, Noor, et al. “Fractal Terrain“, Procedural Content Generation in Games, pp. 62-64, 2016.

[2] Greuter, Stefan, et al. “Undiscovered worlds–towards a framework for real-time procedural world generation.“ Fifth International Digital Arts and Culture Conference, Melbourne, Australia. Vol. 5. 2003

[3] G. Kelly, H. McCabe “Road Network: Template Based Generation.“ ITB Journal 14.3 (2006): 342-351.

[4] Glass, Kevin R., Chantelle Morkel, and Shaun D. Bangay. “Duplicating road patterns in south african informal settlements using procedural techniques.“ Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa. 2006.

[5] Müller, Pascal, et al. “Procedural Urban Modeling in Practice“. 2008

[6] Greuter, Stefan, et al. “Real-time procedural generation of ‘pseudo infinite’ cities“. 2003.

[7] Y. Parish, P. Müller, “Modeling of buildings“, Procedural modeling of cities. 2001

[8] R.M. Smelik, T. Tutenel, K.J. de Kraker, R. Bidarra, “A declarative approach to procedural modeling of virtual worlds“, Computers & Graphics, Volume 35, Issue 2, 2011, Pages 352-363, ISSN 0097-8493,

[9] Müller, Pascal, et al. “Procedural modeling of buildings“. 2006