BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2026
http://excel.fit.vutbr.cz

Procedural Generator of 3D Voxel Maps

Otakar Kočí*

Abstract
This project aims to design a pure procedural generator for large-scale voxel environments and implement it in a sandbox voxel game demo. Its main focus is the generation of natural environments. The generator uses several procedural content generation techniques to assemble the environment, creating terrain and underground, placing water features, coloring biomes, and finally populating it with vegetation and decorations. All necessary voxel models are generated as well. The implemented real-time demo allows visualization, exploration, and editing of the generated environments, and, thanks to a dynamic, chunk-based generation and unloading system, it can support very large worlds. The resulting demo could be used for educational purposes or serve as a backbone for game development.

*xkocio00@stud.fit.vut.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

The use of procedural content generation (PCG) in games and their development has many benefits. One of the most well-known examples is the game Minecraft [1], released in 2011. A sandbox voxel game that unleashes the creativity of its players as well as encourages the exploration of practically infinite, procedurally generated worlds.

This project follows the footsteps of games such as Minecraft and its relatives, aiming to design a procedural generator of natural landscapes that closely reflects real-world properties and implement it in a large-scale voxel demo, which will allow exploration of the generated environment and its basic modification.

The majority of games that use PCG for large-scale voxel environments do not feature very detailed voxels, with sizes around 1 meter. In this sense, this work is inspired by the indie game Lay of the Land[2], aiming to combine its small voxel sizes with the scale of games like Minecraft.

Another thing this project aims to do differently is to use only assets generated at runtime by procedural techniques.

The result is a unique demo featuring fully procedurally generated, nature-inspired voxel environments with high levels of detail. It also supports large-scale worlds and offers basic gameplay features of sandbox voxel games, such as exploration and environment editing.

[1] www.minecraft.net/en-us/about-minecraft
[2] store.steampowered.com/app/2776090

2. World Generator

World generation involves many steps, as shown in **Figure 1**, and uses several PCG methods. A visualization of the process is shown in **Figure 2**.

The initial step in the world generation process is the creation of voxel models for vegetation and decorations. For the generation of trees and bushes, the Space Colonization algorithm is used, which is well-suited to the creation of natural branching patterns [1]. The remaining models (plants, flowers, stalactites, rocks, etc.) use custom solutions.

The generation of the environments begins with the creation of the terrain surface and underground caves, for which 2D and 3D procedural noise are used, respectively. The underground caves are trivially populated with decorations based on random numbers and predefined water levels, whereas the terrain surface is considerably more complex.

Its next step is to extract lakes from the terrain height map using the Priority Flood algorithm, which identifies terrain depressions and flags them as lakes [2]. Then rivers are generated in the opposite direction of their flow, from lakes uphill the terrain, based on terrain gradients.

Lastly, the environment is classified into biomes based

on terrain height, water availability, and environmental age simulated by procedural noise. Biomes influence the terrain's color and the placement of vegetation, decorations, and snow. For example, mountainous areas have conifers and snow, arid areas are covered by sand or rocks, and areas with abundant water are covered by lush vegetation.

3. Demo Implementation

To support truly large environments, the world needs to be split into sections, and only small parts of it can be active at a time. The demo implements this using a dynamic grid of regions and chunks, which serve as an internal representation of the world data.

As the camera moves through the world, its position on the grid is updated, and tasks are scheduled to generate or display new regions and chunks, and unload old ones. The world segmentation is shown in **Figure 4**.

The main pipeline of the demo is visualized in **Figure 3**. The world generator is implemented to produce large regions of mostly 2D information. A larger region size balances out the negative effects of chunking on water network generation.

Then the world data is voxelized into smaller columns of chunks. A voxel representation allows complete editing without compromises and streamlines cave generation implementation.

Lastly, the surface geometry is extracted from the voxel representation and used to visualize the world with chunks of geometry, using the general GPU pipeline.

Environment editing is implemented using ray casts into the voxel structure to locate the specific positions that need to be updated. Affected chunks have their geometry re-generated, and the displayed geometry is swapped. Edit operations are also saved and reapplied when a given world is reopened.

The demo also supports basic level-of-detail mechanisms, player teleportation to specific locations, and a UI for configuring world generation parameters.

4. Generator Capabilities

The generator's capabilities are shown in **Figure 5**. The use of realistic attributes, such as terrain height, water availability, and environmental age, allows the creation of plausible natural environments.

The generator can create a wide range of diverse environments that could encourage exploration, including mountain ranges, deserts, snowy forests, lush meadows, and more.

The caves are highly diverse as well, and further extend the exploration possibilities. This is mostly due to the 3D procedural noise used, as the decoration elements are rather simplistic and could be improved.

Acknowledgements

I would like to thank my supervisor Ing. Michal Vlnas for his valuable help and advice given throughout the countless consultations.

References

[1] Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz. Modeling trees with a space colonization algorithm. In *Proceedings of the Third Eurographics Conference on Natural Phenomena*, NPH'07, page 63–70, Goslar, DEU, 2007. Eurographics Association.

[2] Richard Barnes, Clarence Lehman, and David Mulla. Priority-flood: An optimal depression-filling and watershed-labeling algorithm for digital elevation models. *Computers; Geosciences*, 62:117–127, 01 2014.