FRI-1.414-MIP-04

3D TERRAIN GENERATION SUBSYSTEM[4]

Pr. Assist. Prof. Valentin Velikov, PhD
Department of Informatics and Information Technologies,
University of Ruse “Angel Kanchev”
Tel.: +359 886 011 544
E-mail: vvelikov@ami.un-ruse.bg

Salih Redjeb, Student
Department of Computer Science,
University of Ruse “Angel Kanchev”
Tel.: +359 896 122 807
E-mail: salih.redjeb@gmail.com

Abstract: *The article discusses a newly created subsystem for generating 3D terrain. Existing systems, their positive features and disadvantages are described. The need for the development of one's own is determined.*

There are described the used technologies and algorithms - existing ones, recreated and new ones developed. The adapted mathematical apparatus for 3D modeling, pathfinding in space, coloring and terrain editing is described. As a result, it wqas created an original development (web-based client-server application), which can be used in creating games, virtual reality systems, virtual worlds, virtual routes into existing maps and objects, simulation of natural phenomenas and cataclysms with subsequent training of rescue teams, etc.

The generated terrain/world can be exported from the environment and imported into another system where additional functionalities can be added to it.

Keywords: 3D terrain generation, *Software Engineering, Information systems.*

INTRODUCTION

3D terrain generation is relevant for applications in various fields: such as games, virtual reality, simulations, geographic information systems, architecture, and others.

Such a subsystem - for generating 3D terrain - can be built in different ways, but the main goal is to generate realistic terrain that meets the requirements of the specific application. This can be achieved by using various algorithms and methods, such as Perlin noise, Fractal noise, Voronoi diagrams, etc. To achieve a realistic terrain, it is important to take into account many factors, such as the topography of the area, the type of soil, various natural objects (mountains, rivers, forests and others). Therefore, it is necessary to use appropriate algorithms and methods for terrain generation. In addition to terrain generation, the subsystem can have other functions, such as terrain optimization for better real-time performance, texture generation, adding various objects such as buildings, trees, animals, and others. To achieve better performance and efficiency of the subsystem, various optimization techniques can be used - dynamic leveling of terrain details, use of standardized data formats, and others.

At the beginning of the project, it is important to conduct a detailed analysis of the application requirements and to define the specific functionalities that must be implemented. This may include requirements for terrain visualization, terrain manipulation, interaction with other objects in the scene, and more. Once the application requirements are defined, the appropriate set of algorithms and methods to be used to generate 3D terrain should be selected. This may include noise terrain generation algorithms, realistic texture generation methods, and more. Once the appropriate algorithms and methods have been selected, the software architecture of the 3D terrain generation subsystem must be designed and implemented. This may include creating terrain generation modules, terrain visualization modules, and terrain manipulation modules.

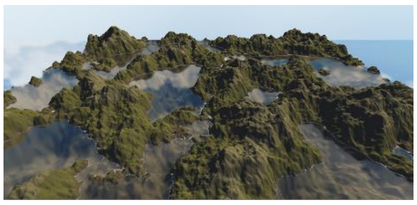
[4] Докладът е представен на конференция на Русенския университет на 27 октомври 2023 г. в секция „Математика, информатика и физика“.

EXPOSITION

1. Overview of existing systems

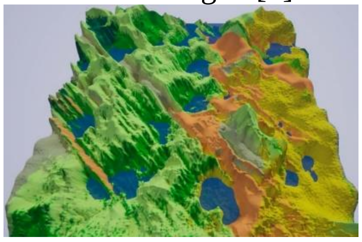
There are many 3D terrain generation systems of similar capabilities aimed at different needs and applications. Some of the most popular ones on the market right now are:

Unity[1]



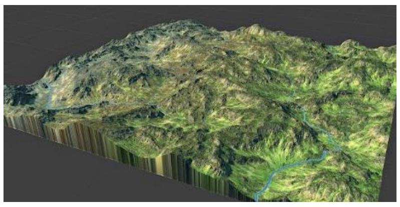
Фигура 1 - Unity терен

Unreal Engine[2]



Фигура 2 - Unreal Engine терен

World Machine[3]

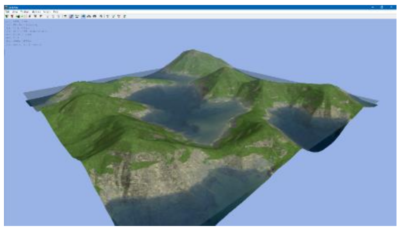


Фигура 3 - World Machine терен

Terragen[4]

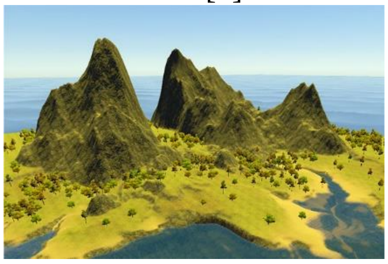
Фигура 4 - Terragen терен

L3DT[5]



Фигура 5 - L3DT терен

Gaia[6]



Фигура 6 - Giaia терен

Comparison of the considered systems (Table 1):

Table 1 - Comparison of the considered systems

system	price	graphical interface	used methods	supported platforms
Unity	free/paid	Yes	Perlin noise, Vorni diagrams, Heightmaps	Windows,Mac, Linux,IOS,Android
Unreal Engine	free/paid	Yes	Simplex noise, Vorni diagrams, Perlin noise, Erosin simulation	Windows,Mac, Linux,IOS,Android
World Machine	paid	Yes	Perlin noise, Erosion simulation, Fractal noise,Hydraulic erosion	Windows
Terragen	paid	Yes	Perlin noise, Hydraulic erosion Erosion simulation,	Windows,Mac
L3DT	free/paid	Yes	Perlin noise, Erosion simulation, Fractal noise, Hydraulic erosion	Windows
Gaia	paid	Yes	Perlin noise, Erosion simulation, Vorni diagrams, Hydraulic erosion	Windows

All the discussed systems have certain characteristics that make them effective for generating 3D terrains. For example, Unity and Unreal Engine are good for generating terrains for PC games and virtual reality, offering a lot of options for setting up and detailing the terrain. WorldMachine and Terragen are very suitable for creating realistic geographical formations, offering different tools to mix different types of noises and filters to achieve the desired results. L3DT is particularly useful for generating large terrains with a lot of detail, offering many options for setting up and controlling the terrain. Finally, Gaia is a good choice for generating terrains for Unity, offering easy integration with this game platform and many options for setting up the terrain. Therefore, for a program to be a good terrain generator, it must have many terrain tuning and control options, as well as be able to generate different types of noise and apply different filters. It is also important to have options for layered texturing of the terrain, allowing for the addition of various elements such as vegetation, rocks, water surfaces and more. In addition, it is important that the program can create high-detail terrains, as well as be able to generate large terrains with small details, offering performance optimization options.

2. System design, technologies used

2.1. Used technologies

In order to design the 3D terrain generation system, one must have a clear idea of the technologies used and their functionality. In this case HTML, CSS, Scss, Javascript, THREE js, Vite and React are used. HTML, Css, Scss are not key to generating the terrain, but are used to create the user interface. Javascript, Three js and Vite are essential.

2.1.1. Javascript

JavaScript [7] is an interpreted language, which gives it great flexibility, this makes it suitable for different types of tasks such as user interaction, data validation, building animations and many more. Today, it is widely used for both web and mobile application development and desktop software, such as online games, social networks, job search applications, and many others.

2.1.2. Three.js

Three.js is a JavaScript library used to create interactive 3D graphics in web browsers. It is based on WebGL, which is a low-level API (Application Programming Interface) for rendering 3D graphics in web browsers [8].

Three.js functionality will be used to render three-dimensional objects, add light in space, add object colors, camera and basic camera movements, add background texture.

2.1.3. Vite

Vite is a modern JavaScript bundler and dev server that is used to create modern web applications. A **bundler** is a tool that bundles different files into one place, usually into a single file that can be used by browsers. This includes JavaScript files, CSS files, HTML files, images and other resources used in the web application [9].

Vite also has a built-in **dev server** that allows developers to see the changes they make to the web application in real-time without the need to restart the server or reload the web page.

2.2. System design

2.2.1. Creating a scene

The first thing to do is to initialize the three js scene. In Three.js, a Scene is an object that represents the 3D space in which all objects and graphics are placed and manipulated. This can include geometric shapes such as cubes, spheres, cylinders and more, as well as lights, cameras, materials, textures and other elements. The scene is responsible for rendering all objects and elements in 3D space. It contains all the objects that need to be visualized and manages them so that a visual view of them can be provided.

After initializing the scene, three js renderer and camera should be created. In Three.js, the Renderer and the Camera are two of the most important components used to render 3D objects and scenes.

The renderer is responsible for generating an image on the screen that is rendered based on the 3D scene and camera. The renderer can be configured to support various rendering options, such as lighting settings and other effects.

The camera in Three.js is a component that determines where in 3D space the viewer is. This includes settings such as camera position, viewpoint angle, size, and more. All objects in the scene are rendered relative to the camera position, allowing the viewer to see different parts of the scene as the camera moves or rotates.

The Three.js renderer and camera work together to generate screen images based on the 3D scene. The renderer uses the information from the camera to calculate how to render the scene and generate the image that is rendered on the screen.

Next is setting the height and width of the render, initially small values will be set (e.g. height and width - 500 pixels each) in order not to load the processor, and finally, after all the settings - the dimensions can be increased to the required ones. Next is a scene illumination (eg: in Y: 100 units, and it points to the center of the scene). The background so far is black, which is not always convenient - the solution is to add a texture (provided by Three js). (fig.7)

2.2.2. Create a plane

Next is a choice of way to create a plane in three-dimensional space. For this, an appropriate data structure in Three.js must be chosen to do the job efficiently. Several different data structures can be used:

- PlaneGeometry - it is a geometric shape that creates a plane in 3D space. This geometry can be used to create a plane with a specified width and height.
- BufferGeometry - it is a lower level of abstraction that allows faster generation of geometry by using data arrays. This geometry can be used to create a plane with a specified width and height.
- Mesh - it is an object that can be used to represent geometry in 3D space. It can include a plane that is created using PlaneGeometry or BufferGeometry.
- PlaneBufferGeometry - it is an optimized version of PlaneGeometry that uses data arrays to achieve better performance.

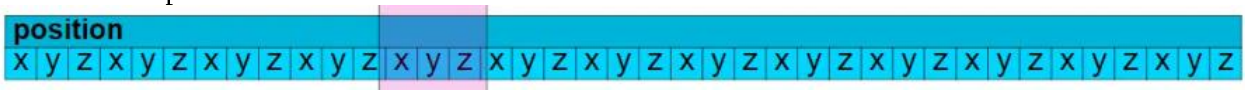

position
x y z x y z x y z x y z x y z x y z x y z x y z x y z x y z x y z x y z x y z

Fig. 8 – BufferGeometry array

BufferGeometry is preferred because the system needs to work quickly with a large number of triangle-polygons. Two constants (**mapWidth** and **mapHeight**) define the size of the plane in three-dimensional space. It will be broken up into small square areas for plane generation. Then the points on the surface are generated, the distance between the points on the plane in X and Y direction is calculated. A positions array (fig.8) is generated, containing the X, Y and Z coordinates of all points on the plane (until now Y=0). Triangles are generated (Fig. 9) that will form the plane (Fig.10).

After all the points and triangles have been generated, the attributes of the buffer geometry are set. This is done through the setIndex and setAttribute methods, which set the point indices and their coordinates respectively.

The end result of this code is a buffer geometry that represents a surface with dimensions mapWidth x mapHeight and with width x height points (Fig. 10).

Fig. 7 - Scene

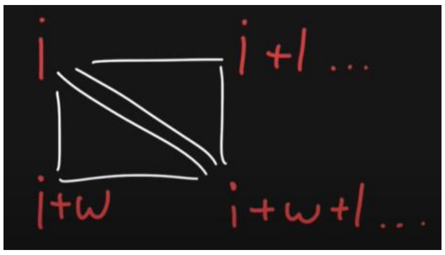


Fig. 8 - Treangles

Fig. 9 - Plane

2.2.3. Perlin noise

After a plane is generated, terrain should be generated on top of it (i.e. – bumps and depressions in the plane). Several ways are possible, Perlin noise will be used.

The problems with random noise [10] are the sudden changes in values. This results in very jagged and unrealistic terrain. And here Perlin-noise comes to the rescue.

Perlin noise is a type of gradient noise developed by Ken Perlin in 1983. It has many applications, such as: procedurally generating terrain, applying pseudo-random changes to a variable, and helping to create image textures. It is most often applied in two, three, or four dimensions, but can be defined for any number of dimensions.

Perlin noise is a procedural texture primitive, a type of gradient noise used by visual effects artists to increase the realism of computer graphics. The feature has a pseudo-random appearance, but all its visual details are the same size. This property allows it to be easily controlled; multiple scaled copies of Perlin noise can be inserted into mathematical expressions to create a wide variety of procedural textures. Synthetic textures using Perlin noise are often used in CGI to make computer-generated visuals – such as object surfaces, fire, smoke or clouds – appear more natural by mimicking the controlled random occurrence of textures in nature [11].

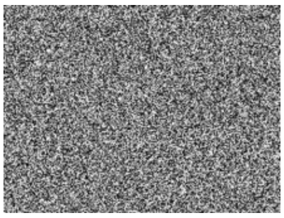


FIg. 10 – random noise

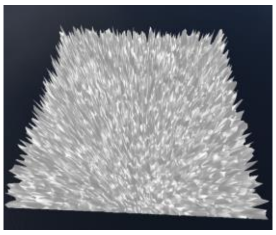


FIg. 11 – random noise - plane

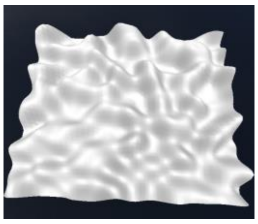


Fig. 12 – Perlin noise - plane

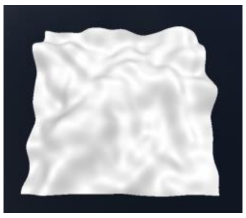


Fig. 13 - Three octaves

A 5-argument function is developed that takes a number from a given range and converts it to a number from another range. It will use 2 more helper functions: the first will normalize the number in a given range, and the second will calculate the value in the new range. This is what the plane looks like (Fig. 12) if for each point the Y coordinate is shifted according to a value given by random noise with a range of 0-1 and is converted to a range of 0-20.

2.2.4. Adding octaves

Octaves in Perlin noise terrain generation are used to add detail and realism to the generated terrain. Perlin noise is generated as a combination of noises at different frequencies (different "octaves").

Each octave includes noise with a higher frequency than the previous one and a smaller amplitude. When the octaves are combined, the higher frequencies of the noise contribute to adding detail and fine detail to the generated terrain, while the lower frequencies give large contours to the generated terrain (Fig. 15).

2.2.5. Coloring the terrain

Terrain visualization can be improved by using values in the range 0 to 1 returned by Perlin noise. Low values can be associated with water features and high values with hills, mountains and other terrain types.

After applying the colors in the respective ranges, the following result is obtained (Fig. 20). However, the boundaries between different colors are too sharp. One solution to this problem is linear interpolation – a new one is generated between 2 adjacent points, for which a new color is calculated and determined (fig. 21).

Example values embedded in the system (Table 2):

Table 2 – Example values for the coloring ranges

Value	Color
0 - 0.15	Blue
0.15 – 0.2	Yellow
0.2 – 0.25	Light Green
0.25 – 0.4	Dark Green
0.4 – 0.6	Light Brown
0.6 – 0.75	Dark Brown
0.75-1	White

The water level should also be leveled - in the event that the relief in the water is not desirable. The change is easy - when generating the Perlin noise if the noise level is below 0.15 it is set to 0.15 (Fig. 23).

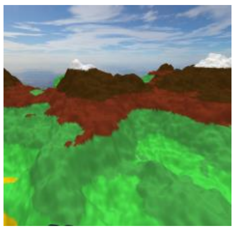


Fig.14 - colors without a smooth transition

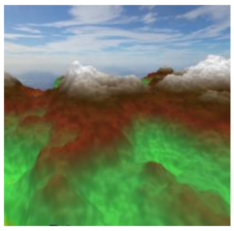


Fig.15 colors with a smooth transition

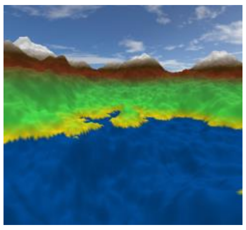


Fig.16 - uneven water

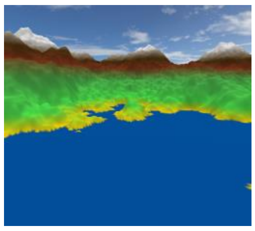


Fig.17 - leveled water

2.2.6. Height map

A height map is a type of graphic that is a two-dimensional black-and-white or color image that is used to determine the height of points in three-dimensional space. Each point on the height map corresponds to a point in the three-dimensional space, and the color or brightness of the corresponding point in the picture determines the height of the point in the three-dimensional space [12]. The steps are similar to the previous ones (BufferGeometry, HTML canvas; after attaching the image to the canvas context, **getImageData** returns an array with information about the ARGB values for the pixels).

2.2.7. Finding the shortest path

To implement finding the shortest path, The Dijkstra's algorithm for finding the shortest path between two points is taken and implemented.

Dijkstra's algorithm is one of the most popular algorithms for finding the shortest path in a graph with non-negative edge weights. It is an iterative algorithm that works by moving from the start vertex to the end vertex, selecting the node with the smallest distance from the start vertex and updating the distances of all its neighbors. This is repeated until the final vertex is reached or until all nodes are traversed [13]. The Dijkstra algorithm is suitable because it works with edge weights. This is useful in this case, as the system will label high areas as mountains and therefore more difficult to traverse. The function implementing the Dijkstra algorithm works according to the following algorithm:

1. The starting point is set, which is known to be at a distance of 0 from itself.
2. For every other vertex, two variables are initialized - the distance from the starting point and the previous vertex with values infinity and undefined respectively.
3. The starting point is placed in a list that will contain the vertices to be checked.
4. The main loop begins, which runs until the list is empty.
5. The vertex with the smallest distance is extracted.
6. For each neighbouring vertex of the current one, which is not marked as visited and is not an obstacle, the distance to the starting vertex is calculated. If this distance is less than the current distance to the neighbour, then the distance and the previous vertex of the neighbour are updated, and the neighbour vertex is added to the list to be checked later.
7. The current vertex is marked as visited.
8. The current peak is removed from the list.
9. After the loop is finished, the shortest path to the set point is returned in the list of all vertices.

After finding the shortest path, a red line is drawn slightly higher than the terrain so that it can be easily distinguished by the user.

2.2.8. Terrain modelling

This functionality enables manual modelling of the terrain - raising or lowering circular areas using the left or right mouse button. The radius and height can be adjusted.

2.2.9. Creation of the graphical interface of the system

Used technologies are HTML, Css/Scss, React, Redux.

The generated terrain can be converted to a .glb or .gltf format file, which can then be downloaded to a local drive. These are two supported and used file formats that work on the major and well-known programs for modelling and working with 3D models.

CONCLUSION

Testing of the system has shown that the terrain it generates has a high level of detail and is able to reproduce a variety of landforms, including mountains, valleys, and more. Able to produce terrains that meet the needs of various projects and applications.

It is recommended when using the system not to modify or change settings or colors on a large number of points making up the terrain. The large number of points puts a strain on the system and makes changing settings and mods slower and less user-friendly. It is recommended to change the settings when the number of points is low and as a final change to increase the number of points. Possible applications of the generated terrain: games, virtual reality, simulation of natural phenomena, etc.

Ideas for further development:

- Creation of a library of blanks (primitives) that can be used to build the terrains: caves, waterfalls, rivers, paths, forests, trees, stumps, cars, ships, planes, various animals, etc.
- Import of 2D maps to generate 3D terrains: these maps can be in different formats such as .jpg and loaded into the system. Once the map is imported, the system can analyse the different colors and their values to determine the terrain height and generate a 3D model that matches the input map. This will allow users to create 3D terrains based on real geographic or physical data.

ACKNOWLEDGMENTS

This publication reflects research from the scientific project 23-FPNO-02 “Investigation of effective knowledge management mechanisms applied in software engineering when creating projects with Agile methodologies” - of the “Scientific Research” fund of Ruse University “Angel Kanchev”, 2023.

REFERENCES

[1] Unity Manual, Unity - Manual: World building (unity3d.com), 15.02.2023, https://docs.unity3d.com/Manual/CreatingEnvironments.html

[2] Unreal Engine Documentation, Landscape Outdoor Terrain | Unreal Engine 4.27 Documentation, 15.02.2023, https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Landscape/

[3] World Machine Features, World Machine Features for Terrain Generation (world-machine.com), 15.02.2023, https://www.world-machine.com/features.php

[4] Terragen, Feature Tour – Planetside Software, 15.02.2023, https://planetside.co.uk/terragen-feature-tour/

[5] L3DT documentation, l3dt:userguide [BundyDocs] (bundysoft.com), 16.02.2023, http://www.bundysoft.com/docs/doku.php?id=l3dt:userguide

[6] Gaia, Gaia Pro - Terrain And Scene Generator | Procedural Worlds (procedural-worlds.com), 17.02.2023, https://www.procedural-worlds.com/products/professional/gaia-pro/

[7] Javascript, JavaScript - Wikipedia, 20.02.2023, https://en.wikipedia.org/wiki/JavaScript#:~:text=JavaScript%20(%2F%CB%88d%CA%92%C9%91%CB%90v,often%20incorporating%20third%2Dparty%20libraries.

[8] Three.js, Three.js - Wikipedia, 20.02.2023, https://en.wikipedia.org/wiki/Three.js

[9] Vite.js, Why Vite | Vite (vitejs.dev), 20.02.2023, https://vitejs.dev/guide/why.html

[10] White noise - Wikipedia, 23.02.2023, https://en.wikipedia.org/wiki/White_noise

[11] Perlin noise - Wikipedia, 23.02.2023, https://en.wikipedia.org/wiki/Perlin_noise#:~:text=Perlin%20noise%20is%20a%20procedural,details%20are%20the%20same%20size.

[12] Heightmap - Wikipedia, 25.02.2023, https://en.wikipedia.org/wiki/Heightmap

[13] DDijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction (freecodecamp.org), 27.02.2023, https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/