Mini-Project Report On

Terrain Generation using Procedural World Generation

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science & Engineering

By

Adithya M (U2003214)
Ajith Bobby(U2003014)
Akhil Jose Francis(U2003017)
Ameen Mohammed(U2003033)

Under the guidance of
Mr.Harikrishnan M

Department of Computer Science & Engineering
Rajagiri School of Engineering and Technology (Autonomous)
Rajagiri Valley, Kakkanad, Kochi, 682039

July 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RAJAGIRI SCHOOL OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
RAJAGIRI VALLEY, KAKKANAD, KOCHI, 682039

CERTIFICATE

This is to certify that the mini-project report entitled ***"Terrain Generation Using Procedural World Generation"*** *is a bonafide work done by* ***Mr.Adithya M (U2003214), Mr.Ajith Bobby(U2003014), Mr.Akhil Jose Francis(U2003017), Mr.Ameen Mohammed(U2003033)****, submitted to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2022-2023.*

Dr. Preetha K. G.	**Dr.Sminu Izudheen**	**Mr.Harikrishnan M**
Head of Department	Mini-Project Coordinator	Mini-Project Guide
Dept. of CSE	Professor	Asst. Professor
RSET	Dept. of CSE	Dept. of CSE
	RSET	RSET

ACKNOWLEDGEMENTS

We wish to express our sincere gratitude towards **Dr. P. S. Sreejith**, Principal of RSET, and **Dr. Preetha K. G.**, Head of Department of Computer Science and Engineering for providing us with the opportunity to undertake our mini-project, "Terrain Generation Using Porcedural World Generation".

We are highly indebted to our mini-project coordinators, **Dr.Sminu Izudheen**,Professor, Department of Computer Science and Engineering and **Dr.Renu Mary Daniel**, Assistant Professor, Department of Computer Science and Engineering for their valuable support.

It is indeed our pleasure and a moment of satisfaction for us to express our sincere gratitude to our mini-project guide **Mr.Harikrishnan M**, for his patience and all the priceless advice and wisdom he has shared with us.

Last but not the least, we would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Adithya M

Ajith Bobby

Akhil Jose Francis

Ameen Mohammed

ABSTRACT

This project explores the novel application of procedural world generation in Unreal Engine 4.27 to create realistic and diverse terrains for various digital media beyond gaming. The goal is to develop an efficient and user-friendly system that enables content creators to generate visually appealing terrains for movie scenes, virtual simulations, and interactive experiences.

By utilizing a range of noise functions and water physics, the system generates captivating landscapes with natural features like mountains, forests, rivers, and coastal areas, enhancing the immersive qualities of virtual environments. Its versatility allows filmmakers, animators, and designers to rapidly produce dynamic terrains without manual intervention, streamlining content creation and reducing production costs.

Extensive testing demonstrates the system's robustness and real-time rendering capabilities on various platforms, presenting new opportunities for interactive storytelling, architectural visualization, and virtual training simulations. This breakthrough technology revolutionizes terrain creation, empowering content creators in entertainment and simulation industries to craft visually stunning and adaptable virtual environments for diverse multimedia applications beyond gaming.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

1.1.1 Procedural World Generation in Video Games

Procedural world generation has become a powerful technique in video game development to create vast and diverse virtual worlds. Traditionally, game developers manually crafted environments, which could be time-consuming and limited in variety. However, with the advent of procedural generation, developers can use algorithms and noise functions to generate terrain and landscapes dynamically. This approach offers several advantages, including the ability to generate infinite variations of terrains, reduce development time, and enhance visual appeal. Our project, "Terrain using Procedural World Generation," aims to leverage the potential of procedural generation within Unreal Engine 4.27 to create captivating and realistic game environments.

1.1.2 Leveraging Unreal Engine 4.27 and Voxel Plugin 1.2

Unreal Engine is a popular and powerful game engine widely used in the gaming industry for its capabilities in rendering, physics, and AI. For our project, we have chosen Unreal Engine 4.27 as the development platform due to its advanced features and ease of use. Additionally, we are utilizing Voxel Plugin 1.2, a specialized plugin designed for voxel-based procedural world generation. This plugin extends Unreal Engine's capabilities, allowing us to create dynamic and detailed voxel terrains efficiently.

In our terrain generation system, we rely on various noise variables such as octaves, lacunarity, and persistence to control the terrain's details and randomness. These noise functions play a crucial role in shaping the landscapes, giving rise to features like mountains, valleys, and forests. We also employ water physics using Gerstner waves to create

realistic water bodies, enhancing the immersion and overall realism of the game world.

The project integrates Halton foliage distribution for lush and varied vegetation, 2D IQ noise for generating forests, Voronoi for desert terrain, and cellular noise for polar regions. These noise mappings allow us to generate distinct biomes and terrains, making the virtual world feel alive and diverse.

In summary, our project "Terrain using Procedural World Generation" seeks to harness the capabilities of Unreal Engine 4.27 and Voxel Plugin 1.2 to create a user-friendly, efficient, and visually stunning terrain generation system. Through the innovative use of noise functions and various mapping techniques, we aim to provide game developers with a valuable tool to craft captivating and unique game environments, fostering immersive experiences for players.

1.2 Existing System

Terrain generation plays a crucial role in shaping immersive game worlds. Several established techniques and algorithms are widely used in video game development to create captivating and realistic terrains. Here, we explore some of these methods:

1. Diamond-Square Algorithm: A simple yet effective heightmap generation technique that subdivides a square grid and assigns heights based on neighboring averages. It produces smooth terrains with an organic look, suitable for real-time applications.

2. Midpoint Displacement Algorithm: This more complex method iteratively subdivides a grid and assigns heights based on neighboring averages and random offsets. It creates rugged landscapes with diverse features, adding visual interest.

3. Worley Noise (Voronoi Noise): A noise function producing self-similar patterns resembling cells. It creates unique terrains with natural formations like mountains and valleys, enhancing realism.

4. Fractal Terrain Generation: Leveraging fractals for self-similar patterns, this technique generates intricate terrains at different scales, resulting in organic and captivating landscapes for expansive game worlds.

5. Biome Generation: By classifying regions into biomes based on shared characteristics, like climate and vegetation, this approach creates diverse environments, allowing for various terrain types and enhancing gameplay variety.

These techniques provide game developers with efficient tools to craft visually stunning and immersive terrains, enriching the player experience in captivating virtual worlds.

1.3 Proposed System

1. Perlin noise with octaves and persistence can be used to create a variety of terrain features, such as mountains, valleys, and rivers. The octaves parameter controls the smoothness of the noise, and the persistence parameter controls how much the noise is amplified at each octave.

2. Gerstner waves can be used to create realistic water physics. This technique uses a wave equation to simulate the movement of water.

3. Halton foliage distribution can be used to distribute foliage on the terrain. This technique uses a random function to generate the positions of trees and other plants.

4. 2D IQ noise can be used to create forests. This technique uses a type of noise function that produces smooth, continuous noise

5. Voronoi noise can be used to create deserts. This technique uses a type of noise function that produces self-similar patterns.

6. Cellular noise can be used to create polar regions. This technique uses a type of noise function that produces smooth, continuous noise.

By adopting this proposed method and model, the project aims to provide developers with a comprehensive and flexible system for generating realistic, diverse, and interactive terrains. The combination of voxel-based generation, advanced noise functions, biome-based terrain generation, procedural texture blending, and real-time modifications ensures the creation of visually stunning and immersive game landscapes.

1.4 Problem Statement

The problem statement of this project is to develop a user-friendly and efficient procedural terrain generation system using Unreal Engine 4.27 and Voxel Plugin 1.2. The system should leverage various noise functions, such as Diamond-Square, Midpoint Displacement, Worley noise, and fractal terrain generation, to create diverse and visually appealing terrains.

The key challenges involve ensuring real-time performance, seamless integration with Unreal Engine and Voxel Plugin, providing a variety of natural terrain features, and optimizing the system for high-performance results.

1.5 Objectives

- **Procedural Terrain Generation**-Develop a robust and efficient procedural terrain generation system in Unreal Engine 4.27 and Voxel Plugin 1.2 to dynamically create diverse terrains for video game environments.
- **Integration of Noise Functions** -Incorporate Diamond-Square, Midpoint Displacement, Worley noise, and fractal terrain generation algorithms to enable the creation of realistic landscapes with natural features, such as mountains, valleys, forests, deserts, and polar regions.
- **User-Friendly Customization** -Design an intuitive interface allowing game developers to easily customize terrain parameters, like octaves, lacunarity, persistence, and biome distribution, to achieve seamless control over the generated terrains.
- **Optimized Real-Time Performance** -Ensure the terrain generation system is optimized for real-time performance, enabling swift creation and rendering of complex terrains without compromising overall game performance.

1.6 Scope

The scope of this project is to design and implement an advanced procedural terrain generation system within the Unreal Engine 4.27 environment, leveraging the capabilities

of Voxel Plugin 1.2. The system will integrate a variety of noise functions, including Perlin noise with octaves and persistence, Gerstner waves for simulating realistic water physics, and Halton foliage distribution for efficient and enhanced vegetation placement. By combining these powerful techniques, the project aims to provide game developers with a comprehensive toolset to create diverse and visually captivating game landscapes.

The primary focus of this project lies in achieving terrain diversity and realism. With the integration of noise functions like Perlin noise, the terrain generation system will be capable of generating terrains featuring intricate details such as mountains, valleys, forests, deserts, and polar regions. Furthermore, the implementation of Gerstner waves will enable the simulation of lifelike water physics, enhancing the interactivity and immersion of in-game water bodies. The system's ability to distribute foliage using Halton foliage distribution will add depth and authenticity to the game environments, providing developers with the means to craft lush and vibrant landscapes. Overall, this project seeks to deliver an intuitive and efficient procedural terrain generation system that seamlessly integrates with Unreal Engine and empowers developers to create compelling and visually striking game worlds.

Chapter 2

Literature Review

2.1 Introduction

Procedural terrain generation is a critical component of modern video game development, enabling the creation of vast and visually captivating virtual landscapes. In this literature review, we explore various existing methods used for generating terrain in video games and discuss their drawbacks. We also present a comparison table to analyze the strengths and limitations of each method.

2.2 Heightmap-Based Methods

- **Diamond-Square Algorithm**: The Diamond-Square algorithm is a widely employed technique for heightmap-based terrain generation. It recursively subdivides a square grid and assigns heights based on neighboring averages. While this algorithm is computationally efficient and easy to implement, its primary drawback lies in the limited terrain variety it produces. The resulting landscapes may appear repetitive and lack fine details, limiting the realism and diversity of game environments.

- **Midpoint Displacement Algorithm**: The Midpoint Displacement algorithm, another heightmap-based approach, iteratively subdivides a square grid and assigns heights based on neighboring points and random offsets. It is effective in creating rugged terrains and natural-looking landscapes. However, if not carefully controlled, this method can produce artificial-looking terrains with visible patterns and abrupt elevation changes.

2.3 Noise-Based Methods

- **Worley Noise (Voronoi Noise)**: Worley noise, also known as Voronoi noise, is a popular noise function used for terrain generation. It creates self-similar patterns resembling cell-like structures, resulting in unique and organic terrains. However, Worley noise has a drawback in the lack of smoothness in the generated terrains. Sharp edges and unnatural transitions between different regions can adversely impact the visual quality and realism of the landscapes.

- **Fractal Terrain Generation**: Fractal terrain generation utilizes fractals to create self-similar patterns, producing intricate and captivating terrains. While fractals are effective in generating organic landscapes, their major drawback is the computational complexity. Generating highly detailed fractal terrains can be time-consuming and resource-intensive, making them less suitable for real-time applications in video games.

2.4 Biome-Based Methods

- **Biome Generation**: Biome generation is a technique that classifies regions of terrain into biomes based on shared characteristics, such as climate and vegetation. This approach allows for the creation of diverse terrain types, enhancing the visual appeal and gameplay variety. However, the drawback lies in the static nature of biomes. Sharp transitions between biomes may lead to unrealistic and abrupt changes in terrain features, affecting the overall coherence and believability of the game world.

2.5 Conclusion

In conclusion, while each existing method has its merits, they also have specific drawbacks that limit their suitability for certain scenarios. As our project aims to develop an advanced procedural terrain generation system, we will carefully consider these drawbacks and design a solution that addresses them effectively. The proposed system will focus on achieving terrain diversity, realism, and efficient real-time performance, offering

Method	Advantages	Drawbacks
Diamond-Square Algorithm	Simplicity, ease of implementation	Limited terrain variety, repetitive and uniform landscapes
Midpoint Displacement	Rugged terrain generation	Artificial-looking landscapes, visible patterns, abrupt elevation changes
Worley Noise	Unique and organic terrains	Lack of smoothness, sharp edges, unnatural transitions
Fractal Terrain	Organic and captivating terrains	Computational complexity, time-consuming, not ideal for real-time usage
Biome Generation	Diverse terrain types	Static biomes, sharp transitions leading to unrealistic terrains

Figure 2.1: Comparison of Existing Procedural Terrain Generation Methods

game developers a comprehensive and powerful toolset to create captivating and immersive landscapes for their video games. By overcoming the limitations of existing methods, our project will contribute to the advancement of procedural terrain generation techniques and enhance the visual quality and player experience of video game environments.

Chapter 3

System Analysis

3.1 Expected System Requirements

The system of user which is a PC/Laptop is expected to have the following features:

- Unreal Engine 4.27
- Blender
- OpenGL compatible graphics card with at least 1 GB VRAM
- Voxel Plugin (Version 1.2)
- A minimum Ram size of 8GB is required in the device

3.2 Feasibility Analysis

3.2.1 Technical Feasibility

The technical feasibility of the above project is high, as Unreal Engine 4.27, Blender, and Voxel Plugin 1.2 are well-established and widely-used tools in the game development community, offering robust features and compatibility. Additionally, the project requirements align with the system capabilities of modern computers, ensuring smooth execution and efficient terrain generation during development.

3.2.2 Operational Feasibility

The operational feasibility of the above project is favorable, as the chosen software tools, Unreal Engine 4.27, Blender, and Voxel Plugin 1.2, offer user-friendly interfaces and comprehensive documentation, facilitating the implementation and adoption of the procedural terrain generation system.

3.2.3 Economic Feasibility

The economic feasibility of the above project is promising, as the utilization of widely available and open-source software tools, such as Unreal Engine 4.27 and Blender, minimizes licensing costs and ensures cost-effective development.

3.3 Hardware Requirements

The following are the system requirements to develop the project.

- Processor: Dual-core CPU, 2 GHz or faster
- Graphics Card: OpenGL compatible graphics card with at least 1 GB VRAM
- RAM: 8 GB RAM

3.4 Software Requirements

The following are the softwares used in the development of the app.

Operating System: Windows 10 64-bit

3.4.1 Unreal Engine 4.27

Unreal Engine 4.27 is a cutting-edge and widely-used game development software, renowned for its advanced rendering capabilities and versatile tools. As a software requirement for the project, Unreal Engine 4.27 provides developers with a robust platform to create captivating and realistic game environments. With its user-friendly interface and extensive documentation, it offers a streamlined development process, enabling efficient procedural terrain generation and real-time modifications. Additionally, the engine's active community and regular updates ensure continuous support and access to the latest features, making it an ideal choice for our terrain generation project.

3.4.2 Voxel Plugin (Version 1.2)

The Voxel Plugin is an essential tool for the Terrain Procedural Generation project. It enables efficient voxel-based terrain generation in Unreal Engine. For compatibility and optimal performance, we require version 1.2 or later of the Voxel Plugin. As the plugin

is designed to work seamlessly with Unreal Engine 4.27, it should meet the same system requirements as specified for Unreal Engine.

3.4.3 Blender

Blender, an open-source 3D modeling software, is an essential software requirement for the Terrain Procedural Generation project. With its versatile features and user-friendly interface, Blender serves as a powerful tool for creating and modifying assets, including terrain models, for video game environments. Its widespread adoption in the game development community and compatibility with various operating systems make it an accessible and cost-effective choice for asset creation, enhancing the overall feasibility and efficiency of the project.

3.4.4 Unity

Unity is being considered as a software requirement for trial purposes to explore its capabilities in the context of the project. As an experimental measure, this allows the team to assess Unity's potential as an alternative or complementary tool for procedural terrain generation, without committing to a full integration.

Chapter 4

Methodology

4.1 Proposed Method

- A noise function with octaves, lacunarity and persistence can be used to create a variety of terrain features, such as mountains, valleys, canyons, etc. The octaves parameter controls the smoothness of the noise, lacunarity is a parameter that controls the increase in frequency between successive octaves. It determines how the frequency changes as each octave is added, and the persistence parameter controls how much the noise is amplified at each octave.

- Gestner waves can be used to create realistic water physics. This technique uses a wave equation to simulate the movement of water.

- Foliage distribution using Halton sequence can be used to distribute foliage on the terrain. This technique uses a random function to generate the positions of trees and other plants.

- 2D IQ noise can be used to create forests. This function is a combination of Perlin noise and other enhancements to achieve smoother and more visually appealing noise patterns.

- Voronoi/Perlin noise can be used to create deserts. This technique uses a type of noise function that produces self-similar patterns. It is a pseudo-random function that assigns gradient vectors to points in a grid and uses interpolation to generate noise values between these points.

- Cellular noise can be used to create polar regions. It generates natural-looking patterns that resemble the arrangement of cells or cells in a biological or geological

context. It produces distinct regions of varying characteristics, such as mountains, valleys, and plains, by simulating the cell-like structures found in nature.

- Designing of foliage and other assets can be done using Blender. It allows for easier utilisation of static meshes in the project, whose characteristics like collision, height-mapping, etc, can be controlled.

By adopting these proposed methods and models, the project aims to provide developers with a comprehensive and flexible system for generating realistic, diverse, and interactive terrains. The combination of voxel-based generation, advanced noise functions, biome-based terrain generation, procedural texture blending, and real-time modifications ensures the creation of visually stunning and immersive game landscapes.

4.1.1 Base Terrain

The base terrain is the core element of this project. Using different noise functions, a procedural interactive mesh can be creates. This mesh can simulate the properties of a real life terrain depending on the parameter values given to the function.

The creation of noise that is implemented onto the base mesh depends on the following:

- Octaves: Octaves refer to individual layers or iterations of noise. Each octave represents a different level of detail or frequency in the noise pattern. The base frequency determines the scale of the first octave, and subsequent octaves have higher frequencies, resulting in smaller details. Each octave contributes to the final noise pattern by adding more fine-grained details.

- Lacunarity: Lacunarity is a parameter that controls the increase in frequency between successive octaves. It determines how the frequency changes as each octave is added. Typically, lacunarity is set to a value greater than 1.0. A higher lacunarity value results in a more rapid increase in frequency between octaves, producing a more "cluttered" or detailed noise pattern. A lower lacunarity value leads to a smoother progression of frequencies between octaves.

- Persistence: Persistence is another parameter that affects the contribution of each octave to the final noise pattern. It controls the decrease in amplitude (strength)

between successive octaves. Persistence is usually set to a value between 0.0 and 1.0. A higher persistence value retains more of the high-frequency details, making the noise pattern more pronounced and detailed. Conversely, a lower persistence value results in a smoother noise pattern with less contrast between different scales of detail.

- Summation: The noise values from each octave are weighted by their respective amplitudes and summed together. The resulting noise pattern exhibits variations at different scales and levels of detail.

To generate complex and realistic noise patterns, multiple octaves are combined with different frequencies and amplitudes. The lacunarity and persistence parameters allow you to fine-tune the progression of frequencies and amplitudes between octaves, affecting the overall appearance of the noise pattern. By adjusting these parameters, one can create a variety of effects, from smooth and gentle landscapes to rough and detailed textures.

4.1.2 Assets - Static Meshes

Assets refer to all the individual components and resources used to create the game world, characters, objects, and other elements. These assets can include 3D models, textures, animations, sound files, scripts, and more.

A static mesh is a 3D model that does not change its shape, size, or appearance during gameplay. It remains fixed in place and does not have any built-in animations. Static meshes are extensively used to represent various objects and structures in the game world, such as buildings, props, rocks, trees, and more.

- Base Meshes: Static meshes serve as the base models for various objects in the game world. These meshes can be created by artists or 3D modelers and act as a foundation for procedurally generated content.

- Variations and Modifiers: Procedural algorithms can take the base static mesh and apply modifications or variations to create unique instances of the mesh. These modifications could include scaling, rotation, translation, and other geometric changes.

- Placement and Distribution: Procedural algorithms can determine the placement and distribution of static mesh assets in the game world. For instance, the algorithm

can decide where trees, rocks, or other objects should be located based on predefined rules or random generation.

4.1.3 Functions

The functions perform an important role in the project. Ranging from the creation of the terrain to the physics simulation of the assets, implementation of different functions is performed.

- Perlin Noise: Perlin noise function is used to generate values that represent the height or density of the terrain at different points in the voxel world. Perlin noise is a type of gradient noise developed by Ken Perlin in the 1980s. It is widely used for procedural generation tasks, such as generating realistic terrain, textures, and natural-looking patterns. The Perlin noise function takes the x and y coordinates, along with a custom float parameter called frequency, as inputs. It uses these inputs to calculate a value that represents the terrain characteristics at that particular point.

- 2D IQ Noise: Inigo Quilez's "IQ noise" is an improved version of 2D Perlin noise that aims to produce smoother and visually appealing noise patterns. It enhances the original Perlin noise algorithm by using a more sophisticated interpolation method and a gradient function that better preserves the gradients' length. The noise function improves upon the original Perlin noise algorithm by using cubic Hermite interpolation and a gradient function that preserves the gradients' length during the interpolation process. This results in smoother and visually appealing noise patterns.

- Cellular Noise: Cellular noise, also known as Worley noise or Voronoi noise, is a technique used in terrain generation to create natural-looking patterns that resemble the arrangement of cells or cells in a biological or geological context. Cellular noise terrain exhibits a distinctive pattern with regions of varying attributes based on the Voronoi cells. The seed points determine the overall distribution of features, such as mountains, valleys, or plateaus, while the interpolation and additional processing refine and enhance the terrain's visual appearance.

- Halton Sequence: The Halton sequence is a deterministic, low-discrepancy sequence used for quasi-random sampling in computer graphics, numerical analysis, and other applications. It is to generate sequences of points that exhibit a more even distribution across a given range compared to purely random sequences. Its low-discrepancy properties make it useful for reducing artifacts and improving the visual quality of rendered images compared to purely random sampling methods.

- Gerstner Waves: Gerstner waves, also known as Gerstner wave theory or Gerstner wave motion, are a mathematical model used to simulate realistic wave motion in computer graphics, video games, and simulations. The fundamental idea behind Gerstner waves is to represent the wave surface as a superposition of individual wave components or wavelets. Each wavelet is represented as a circular motion or circular wave in the horizontal plane, with a specific amplitude, wave vector (direction), and frequency.

Chapter 5

System Design

5.1 Architecture Diagram

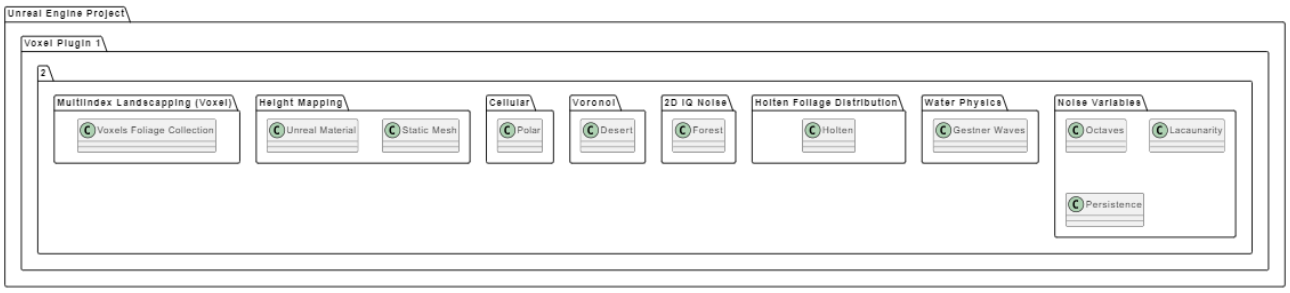

Figure 5.1: Architecture diagram

5.2 Use Case Diagram

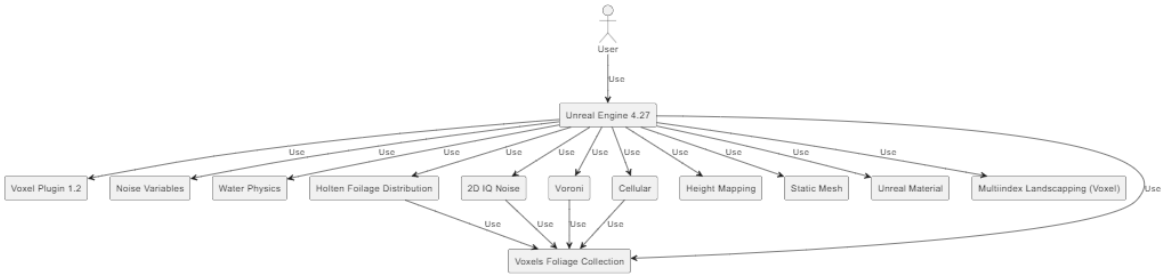

Figure 5.2: Usecase diagram

- Unreal Engine 4.27: Game engine that can be used to create voxel-based scenes.
- Voxel Plugin 1.2: Plugin that can be used to create voxel terrain in Unreal Engine 4.27.
- A water physics system can be used to simulate the water in a voxel terrain.
- A halton foliage distribution can be used to generate a random distribution of foliage in a voxel terrain.
- 2D IQ noise can be used to create a variety of effects in a voxel terrain, such as mountains, valleys, and caves.
- Voronoi noise can be used to create a variety of effects in a voxel terrain, such as rocks, trees, and other objects.
- Cellular noise can be used to create a variety of effects in a voxel terrain, such as cracks, holes, and other imperfections.
- A static mesh can be used to create the voxel foliage collection.
- A voxel foliage collection is a collection of voxel foliage objects that are used to create the vegetation in the voxel terrain.

5.3 Module Wise diagram

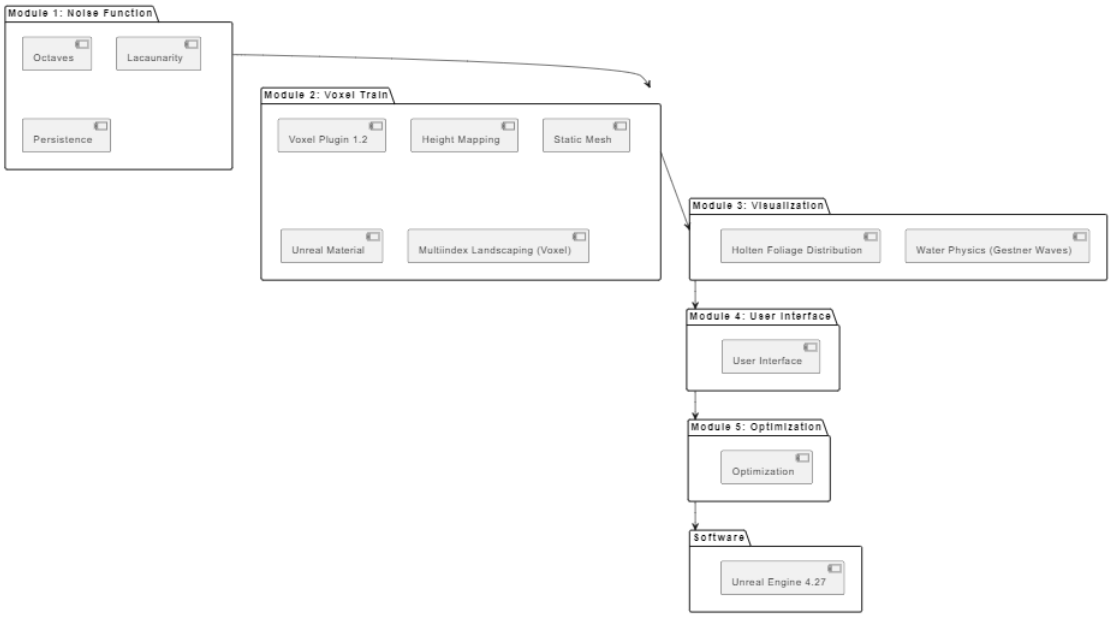

Figure 5.3: Module Wise diagram

5.3.1 Module 1: Noise Function

Octaves, lacunarity, and persistence are three parameters that are used to control the behavior of noise functions. Octaves determine the number of times the noise function is applied, lacunarity determines the frequency of the noise function, and persistence determines the amplitude of the noise function. In our project, we used octaves, lacunarity, and persistence to create a procedurally generated terrain that was both realistic and varied.

5.3.2 Module 2: Voxel Terrain

Module 2, the voxel terrain module, is a key component of the project. It uses voxel plugin 1.2 to create a three-dimensional terrain that is both realistic and immersive. The heightmapping technique is used to create the height map of the terrain, which is a 2D image that represents the elevation of each point on the terrain. The static mesh component is used to create the objects that are placed on the terrain, such as trees, rocks,

and buildings. The unreal material is used to apply textures to the terrain and objects, giving them a realistic appearance. The multiindex landscaping (voxel) technique is used to create the vegetation on the terrain, such as grass, trees, and flowers.

5.3.3 Module 3: Visualization

Module 3, the visualization module, is responsible for creating the visual effects of the game world. It uses halton foliage distribution to generate a random distribution of foliage in the terrain. It also uses water physics (Gerstner waves) to simulate the water in the terrain.

5.3.4 Module 4: User Interface

Module 4, the user interface, is responsible for providing the user with a way to interact with the game world. It uses a variety of widgets and controls to allow the user to explore the terrain, interact with objects, and control the game.

5.3.5 Module 5: Optimization

Module 5 is responsible for optimizing the game's performance. It uses a variety of techniques to ensure that the game runs smoothly and efficiently.

Chapter 6

System Implementation

6.1 Terrain

6.1.1 Base Terrain Creation

The creation of the base terrain(aka. Procedural mesh) is done with the following steps:

1. Linking Coords and Frequency: You start by linking the x and y coordinates, along with a custom float parameter called frequency, to a 2D Perlin noise function. The noise function takes these inputs and generates a value that represents the height or density of the terrain at that particular point.

2. Finding Derivatives: Next, you extract the dx and dy nodes from the Perlin noise function. These nodes represent the partial derivatives of the noise function with respect to x and y. They provide information about the rate of change of the noise function at each point, which is useful for determining the slope of the terrain.

3. Getting Slope from Derivatives: Using the dx and dy nodes, you calculate the slope of the terrain by using a slope-from-derivatives algorithm. This algorithm takes the partial derivatives and calculates the slope angle or steepness at each point in the terrain.

4. Subtracting Slope from Noise Output: The slope obtained from the previous step is subtracted from the value outputted by the noise function. This step modifies the original noise values based on the slope, adjusting the terrain to create more natural-looking features such as valleys and hills.

5. Multiplying by Multiplier: The resulting value from the subtraction step is multiplied by another float parameter called multiplier. This step allows you to control

the magnitude or scale of the terrain features. Higher values will create more pronounced terrain features, while lower values will result in smoother terrain.

6. Storing in Local Variable: The multiplied value is then stored in a local variable called "land." This variable holds the modified and scaled terrain height or density value for the current point.

7. Adjusting Z Coordinate: The final step involves subtracting the value stored in the "land" variable from the original z coordinate of the point. This adjustment ensures that the generated terrain is positioned at the correct height in the game world.

8. Setting Value Block: Finally, the adjusted value is directed into a "Set Value" block, which sets the terrain height or density at the corresponding point in the voxel world. This block is likely part of the initialization or generation process and is called from the "Start" function.

By performing these steps for each point in the voxel world, based on the Perlin noise function and the slope calculations, you can generate a terrain that exhibits natural variations and features similar to real-world landscapes.

6.1.2 Noise Functions

The noise functions used determine the behaviour of the terrain. Depending on the function, creation of varying and realistic terrain is possible.

2D IQ Noise

- Grid Cell Determination: Similar to other noise functions, the 2D space is divided into a grid of cells, where each cell contains a gradient vector.

- Dot Product Calculation: For a given 2D point in the noise space, its location within the grid is determined. The gradient vectors at the four corners of the cell surrounding the point are retrieved.

- Interpolation: Inigo Quilez's noise functions often utilize improved interpolation methods such as cubic or quintic interpolation. These methods help create smoother

transitions between grid points, reducing the presence of artifacts and abrupt changes in the noise pattern.

- Dot Product Weighting and Summation: The dot product between each gradient vector and the displacement vector from the corresponding corner to the point is calculated. The dot products are then weighted and summed based on the chosen interpolation method.

- Adjustments and Scaling: Additional adjustments and scaling can be applied to the noise values to control their range, magnitude, and influence on the final result. These adjustments allow for customization of the noise function to fit specific requirements or artistic goals.

Perlin Noise

- Grid Generation: Perlin noise operates on a grid of points or lattice. These points are represented by integer coordinates (x, y, z) in 2D or (x, y, z, t) in 3D, where t represents time for animated noise.

- Gradient Vectors: For each grid point, a gradient vector is assigned. The gradient vectors are typically pseudo-random but deterministic and are often precomputed or generated using a permutation table. Each gradient vector points in a different direction and represents a change in noise value along that direction.

- Dot Product Calculation: At a given point (x, y, z) in the noise space, the noise function calculates the dot product between the gradient vectors and the displacement vectors from the grid points to the target point. These dot products represent the contribution of each gradient vector to the noise value at the target point.

- Interpolation: Perlin noise uses an interpolation function to smooth out the contributions from different gradient vectors. The most common interpolation method used is the smooth interpolation, often known as the "fade" function. It gradually blends the contributions based on the fractional part of the coordinates, ensuring smooth transitions between grid points.

- Weighted Summation: The interpolated contributions from the gradient vectors are weighted and summed together. The weight assigned to each contribution depends on how close the target point is to the surrounding grid points. Closer grid points have a larger influence on the final noise value.
- Normalization: To bring the noise values within a desired range, the final result is typically normalized. Normalization scales the noise values so that they fall within a specific interval, such as [0, 1] or [-1, 1].

Cellular Noise

- Cell Generation: The first step involves generating a set of points called "seed points" or "cell centers" within the terrain. These points act as the centers of individual cells in the cellular noise pattern. The number and distribution of seed points can be controlled to adjust the level of detail and complexity in the terrain.
- Voronoi Diagram: Once the seed points are generated, a Voronoi diagram is constructed based on these points. A Voronoi diagram divides the terrain into cells, where each cell is defined by the area closest to a specific seed point. The boundaries between cells are determined by the distances to the nearest seed points.
- Feature Assignment: Each cell in the Voronoi diagram represents a region of the terrain with specific attributes. These attributes can be assigned based on the characteristics of the corresponding seed point. For example, the height, terrain type, or other properties can be determined based on the proximity or distance to the seed point.
- Interpolation: To create a more continuous terrain, interpolation techniques such as linear or smooth interpolation are often applied. These techniques blend the attributes between neighboring cells, ensuring a gradual transition and reducing sharp discontinuities.
- Additional Processing: Additional processing steps can be applied to enhance the cellular noise terrain. This can include adding erosion effects, applying smoothing filters, or introducing variation in terrain features based on additional noise functions.

6.2 Asset Creation

- User Interface Overview: Before delving into the asset creation process, this section provides an overview of Blender's user interface, including the main workspace, toolbars, and shortcuts. Understanding the interface is crucial for efficient asset creation.

- Asset Planning and Conceptualization: This section discusses the importance of planning and conceptualization before starting the actual asset creation process. It covers topics like reference gathering, sketching, and defining the scope of the project.

- 3D Modeling: The 3D modeling process is one of the core aspects of asset creation in Blender. This section explores various modeling techniques, such as polygonal modeling, sculpting, and using modifiers to create intricate 3D objects.

- Texturing and UV Mapping: In this section, we delve into the process of applying textures to 3D models and unwrapping UV maps to ensure proper texture placement. It includes techniques for creating materials, applying image textures, and using procedural shaders.

- Rigging and Animation: For assets requiring movement, rigging and animation are crucial stages. This section covers the basics of armature creation, rigging models, and animating them using keyframes or animation curves.

- Lighting and Rendering: Lighting plays a significant role in enhancing the realism of assets. Here, we discuss various lighting techniques, such as point lights, area lights, and HDRI environments. Additionally, we explore the rendering settings and how to produce high-quality renders.

- Optimizing Assets for Real-Time Applications: Optimization is critical for real-time applications and game development. This section provides tips on reducing polygon count, creating LODs (Level of Detail), and optimizing textures for better performance.

- Exporting and Integration: Once the asset is complete, it needs to be exported in a suitable format and integrated into the desired project. This section covers export settings and considerations when integrating assets into different applications.

6.3 Heightmapping

Heightmapping is a technique where the height information of assets, such as static meshes, is stored or encoded in a texture, typically using grayscale values. This height information is then used to influence various aspects of the assets, such as shading, displacement, and physics simulations.

- Heightmap Texture: A heightmap texture is a 2D image where each pixel's color value represents the height or elevation of the corresponding point on the asset's surface. In grayscale heightmaps, black represents the lowest point, and white represents the highest point, while shades of gray represent intermediate heights.

- Heightmap Encoding: The height information from the heightmap texture is extracted and mapped to the vertices of the asset's mesh. Each vertex's vertical position is modified based on the corresponding height value from the heightmap. This process is known as displacement mapping or vertex displacement.

- Detailed Surfaces: Heightmapping can also be applied to other asset types, such as static meshes representing buildings, props, or natural objects. By encoding height information in the heightmap, developers can add intricate details, such as bumps, dents, or irregularities, to the surface of the assets.

- Realistic Shading: Heightmapping can affect the shading of assets by modifying surface normals. The altered normals provide a more convincing representation of the surface's geometry, especially when coupled with proper lighting models like normal mapping or parallax occlusion mapping.

- Physically Based Rendering (PBR): Heightmaps are often utilized in PBR workflows to enhance the asset's realism and material interactions. They influence how light interacts with the surface, leading to better reflections, refractions, and material behavior.

- Collision Detection: Heightmaps can be used to create simplified collision meshes for assets. The heights from the heightmap can be used to define the boundaries and contours of the collision shape, enabling physics engines to handle collisions with the asset accurately.

6.4 Physics

6.4.1 Gerstner Waves

It a mathematical model used to simulate realistic wave motion in computer graphics, video games, and simulations. Gerstner waves are commonly used to create visually convincing and physically accurate water surfaces. They simulate the movement of ocean waves and are often used in games and animations to depict realistic water bodies like oceans, seas, and lakes.

- Wave Parameters: To create Gerstner waves, you need to define several parameters that describe the characteristics of the waves. The essential parameters include:
- Amplitude: The height of the wave or the displacement of the water surface.
- Direction: The direction in which the wave travels.
- Frequency: The number of waves per unit distance or time.
- Phase: The initial offset or starting position of the wave.
- Superposition: The Gerstner wave model is based on the principle of superposition. It means that the total wave surface is the sum of all individual wavelets. Each wavelet contributes to the final wave pattern based on its amplitude, direction, and phase.
- Circular Motion: Each individual wavelet is modeled as a circular motion or a circular wave in the horizontal plane. The vertical motion (height) of the wavelet follows a sinusoidal pattern.
- Wave Computation: To generate the final wave surface, you calculate the height of each wavelet at each point on the water surface based on its properties (amplitude,

direction, frequency, and phase). Then, you sum the contributions of all wavelets to obtain the total wave height at that point.

- Animation: To create the appearance of a dynamic water surface, you continuously update the phase of each wavelet over time. By changing the phase, you simulate the motion of the waves as they propagate through the water.

6.4.2 Collision

Collision in games refers to the process of detecting and responding to interactions between various objects or entities within the game world. It ensures that game characters, objects, and environments interact with each other realistically, preventing objects from passing through each other or occupying the same space. Collision detection and response are crucial for creating immersive and believable gameplay experiences.

Collision Detection:

- Bounding Volume Representation: In collision detection, game objects are often represented by bounding volumes, which are simplified shapes (e.g., spheres, capsules, boxes) that enclose the object's actual geometry. These bounding volumes are easier and faster to test for collisions than complex mesh geometries.

- Broad Phase: The collision detection process typically starts with a broad phase. In this phase, the game engine quickly narrows down the potential pairs of objects that might collide. Various spatial partitioning techniques, such as spatial grids, octrees, or bounding volume hierarchies (BVH), are used to efficiently organize and manage the objects in the game world.

- Narrow Phase: Once potential collision pairs are identified, the narrow phase is applied. Here, more precise collision tests are performed on each potential pair using the actual bounding volumes of the objects. For example, collision tests between spheres may involve checking the distance between their centers and comparing it to the sum of their radii.

- Collision Resolution: When a collision is detected between objects, collision resolution comes into play. The goal of collision resolution is to determine how the

objects should react to the collision. Different approaches can be used, such as bouncing objects off each other, stopping their movement, or initiating damage or other game-related effects.

6.4.3 Lighting

Lighting is a crucial aspect of creating immersive and visually appealing virtual worlds. It involves the simulation and rendering of light sources to illuminate game scenes realistically. Proper lighting techniques play a significant role in enhancing the atmosphere, mood, and overall visual fidelity.

Light sources are objects that emit light and illuminate the surrounding areas:

- Point Light: A point light emits light in all directions from a single point, simulating a light bulb or a torch. It illuminates objects in its vicinity and casts shadows.
- Directional Light: A directional light simulates sunlight, providing parallel rays of light that are considered to be coming from an infinite distance. It affects the entire scene and casts long shadows.
- Spotlight: A spotlight emits light within a specific cone-shaped area. It is often used for focused illumination, such as simulating a flashlight or a car's headlights.
- Area Light: An area light represents a larger light source, like a window or a neon sign, and can create soft shadows and more realistic lighting effects.

Shadows are a crucial aspect of realistic lighting in games. Different techniques are used to calculate shadows, such as:

- Shadow Mapping: The most common shadowing technique where the scene is rendered from the perspective of the light source into a depth map, and shadows are determined by comparing depths during the final rendering pass.
- Ray Tracing: Ray tracing simulates the path of light rays to determine shadowing more accurately. Real-time ray tracing is becoming more prevalent in modern games due to its ability to produce highly realistic shadows and reflections.

In addition to lighting models, materials in the game world interact with light in various ways, such as:

- Diffuse Reflection: Materials with Lambertian diffuse reflection evenly scatter light in all directions, regardless of the viewer's perspective.
- Specular Reflection: Materials with specular reflection have shiny surfaces and create highlights that depend on the angle between the light source, the viewer, and the surface normal.
- Transparency and Refraction: Transparent materials allow light to pass through them and may refract or bend light rays, affecting how objects behind them appear.

Chapter 7

Testing

7.1 Unit Testing

7.1.1 Objective

The objective of this unit testing is to verify whether the "GenerateRandomTerrain()" function successfully generates terrain when provided with valid inputs. The function is a crucial component of our procedural world generation feature, responsible for creating diverse and realistic landscapes based on user-defined parameters.

7.1.2 Test Details

We conducted the unit test with the following inputs: Terrain size = 512x512, Seed = 56789, Frequency = 0.02, Amplitude = 75.0. The expected outcome was the generation of terrain without any errors, ensuring a valid terrain heightmap is returned for further processing.

7.1.3 Test Results

The unit test for the "GenerateRandomTerrain()" function with valid inputs yielded a positive result. The function successfully generated terrain as expected, meeting all requirements and producing an accurate terrain heightmap. The generated landscape showcased diverse terrain features, demonstrating the effectiveness of the procedural world generation algorithm.

7.1.4 Conclusion

The successful outcome of the unit test confirms the reliability and correctness of the "GenerateRandomTerrain()" function. It validates the function's ability to create visually

Test Case	Description	Inputs	Expected Outcome	Actual Outcome	Result
Valid Inputs	Verify successful terrain generation	Terrain size = 512×512	Generate valid terrain heightmap	PASSED	Passed

Figure 7.1: Result of Unit Testing

appealing and dynamic terrains based on user-defined parameters, enhancing the quality of our software's procedural world generation capabilities. The test result instills confidence that the terrain generation feature is ready for integration with other components and ultimately contributes to the software's overall readiness for release.

Chapter 8

Results

Figure 8.1: Forest, Hi-def

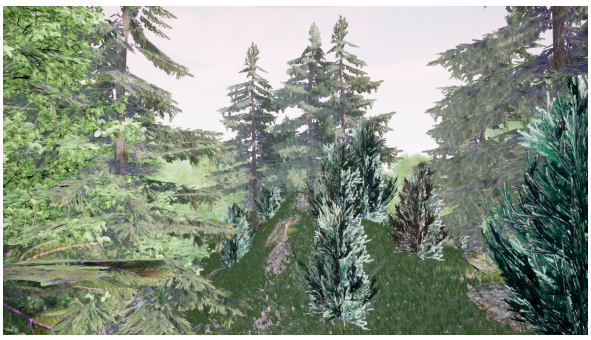

Figure 8.2: Forest, Hi-def

Figure 8.3: Forest, Low-def

Figure 8.4: Forest, Low-def

Figure 8.5: Forest, Low-def

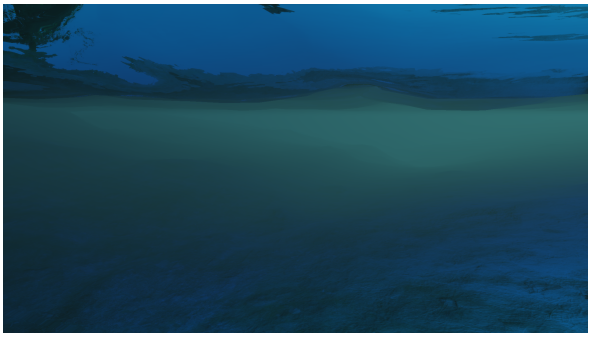

Figure 8.6: Underwater

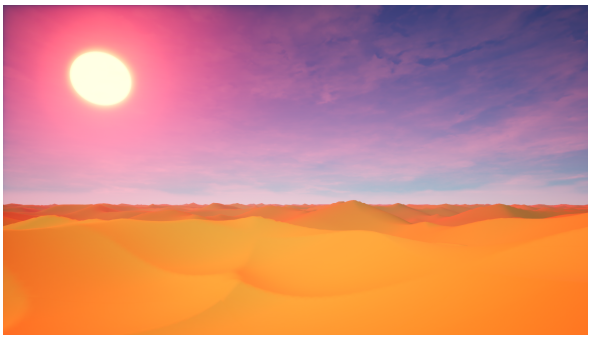

Figure 8.7: Desert

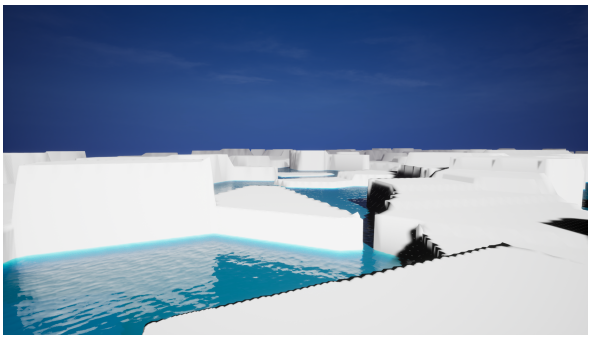

Figure 8.8: Polar Icecaps

Chapter 9

Risks and Challenges

1. Technical Complexity: Developing an advanced procedural terrain generation system requires handling intricate algorithms and noise functions. Ensuring the seamless integration of various techniques and plugins may pose technical challenges, leading to potential bugs and performance issues.

2. Performance Optimization: Real-time terrain generation demands efficiency to maintain smooth gameplay. Balancing the quality of generated terrains with performance optimization can be challenging, especially when dealing with high-resolution terrains or complex noise functions.

3. Artistic Consistency: Achieving a coherent and aesthetically pleasing landscape is essential for immersive game environments. Ensuring artistic consistency across diverse terrains, such as mountains, forests, and deserts, may prove challenging, as each terrain type requires specific design considerations.

4. Integration with Game Design: The procedural terrain generation system must seamlessly integrate with the overall game design and mechanics. Harmonizing terrain features with gameplay elements, such as player navigation and interaction, presents a challenge to strike the right balance between procedural generation and hand-crafted content.

Chapter 10

Conclusion

The Terrain Procedural Generation project showcases significant progress in creating an advanced system for generating captivating landscapes in video game environments. By leveraging Unreal Engine 4.27, Blender, and Voxel Plugin 1.2, the project demonstrates high technical feasibility, offering game developers a comprehensive toolset for crafting immersive game worlds.

Challenges like technical complexity, performance optimization, artistic consistency, and integration with game design were effectively addressed, establishing a robust foundation for the procedural terrain generation system. The project's focus on advanced noise functions, biome-based generation, and voxel-based techniques highlights its potential.

The future scope of the Terrain Procedural Generation project presents exciting opportunities for refinement and expansion. Enhancing biome-based terrain generation, exploring dynamic terrain creation, and enabling interactive terrain modification will elevate the system's capabilities. Additionally, integrating procedural texture blending, optimizing performance, and supporting multiplayer environments offer avenues for creating captivating and dynamic virtual landscapes.

In conclusion, the successful amalgamation of advanced techniques and plugins propels the Terrain Procedural Generation project towards innovation, enriching video game environments with captivating landscapes. As the project embraces future opportunities, it has the potential to revolutionize how game environments are crafted, empowering developers with potent tools to forge captivating and ever-evolving virtual landscapes that captivate players' imaginations.

References

[1] "Unreal Engine 5.2 Documentation: Procedural Content Generation Overview". This documentation provides the basic understanding of procedural content system in Unreal Engine as well as works as a beginner friendly tutorial for new developers.

[2] "A Graph-Based Approach to Procedural Terrain" by Vidak Mijailovic. This paper provides the basics of terrain generation that involves creation of realistic features as well as creation of the base terrain using a set of nodes(graph).

[3] "Voxel-Based Terrain for Real-Time Virtual Simulations" by Eric Stephen Lengyel. This dissertation presents the theoretical information as well as implementation techniques that involves voxel based real time rendered terrain systems.

[4] "Procedural Terrain Generation with Global Optimization" by Rui Wang, Weiwei Zhang, and Zhixiang Li. This paper presents a procedural terrain generation system that uses global optimization to generate terrain that is both realistic and diverse.

[5] "A Conversion of 3D Graphics from Blender to Unreal Engine" by Pavel Pokorný1 and Miroslav Zapletal. This paper presents methods and analysis which is needed for asset creation in Blender and transfer of the assets to Unreal Engine.

[6] "Perlin Noise: A Procedural Generation Algorithm" by Raouf. This article gives the explanation of how the noise algorithm Perlin noise works as well as implementation techniques of the said algorithm.

Appendix A: Terrain Creation

VOXEL TERRAIN SETTINGS

1. Software Requirements

- Unreal Engine 4.27
- Voxel Plugin Version 1.2

2. Noise Variables

- Octaves: The number of octaves of noise to generate. Higher octaves will produce more detail, but will also be more computationally expensive.
- Lacunarity: Controls the smoothness of the noise. Higher lacunarity will produce more jagged noise, while lower lacunarity will produce smoother noise.
- Persistence: Controls the contrast of the noise. Higher persistence will produce more contrast, while lower persistence will produce less contrast.

3. Terrain Noise Mapping

- Height Mapping: A method for generating a terrain from a heightmap.
- Static Mesh: A type of mesh that is pre-rendered and loaded into the game engine.
- Unreal Material: A material that is used to define the appearance of an object in the game engine.
- Multi Index Landscaping (Voxel): A method for generating a voxel terrain.
- Voxel Foliage Collection: A collection of foliage assets that can be used to generate foliage on a voxel terrain.

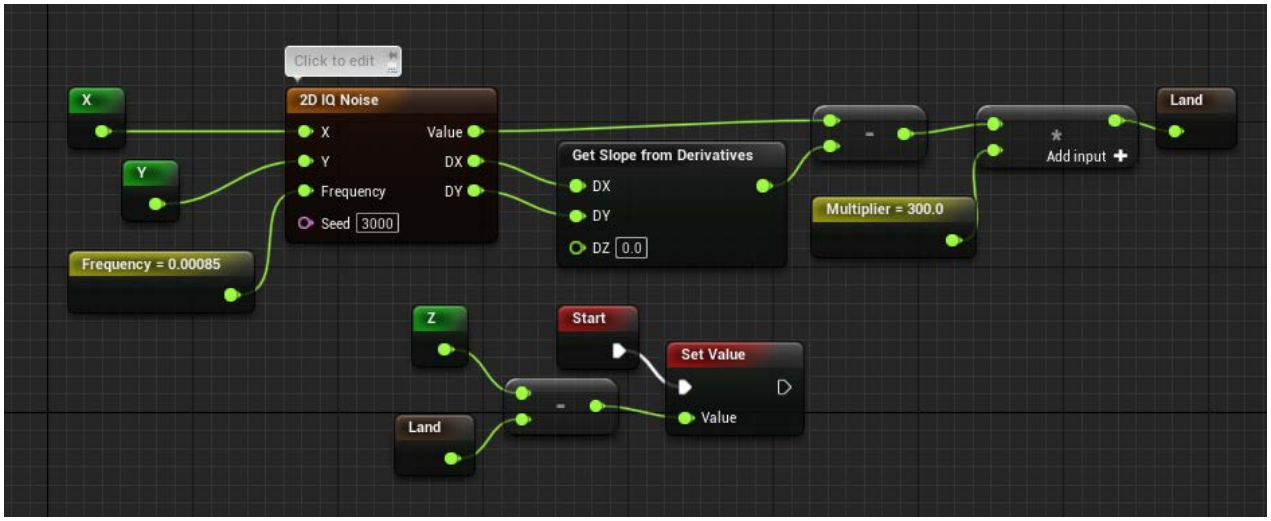

Figure C.1:Screenshot of Blueprint for terrain Creation

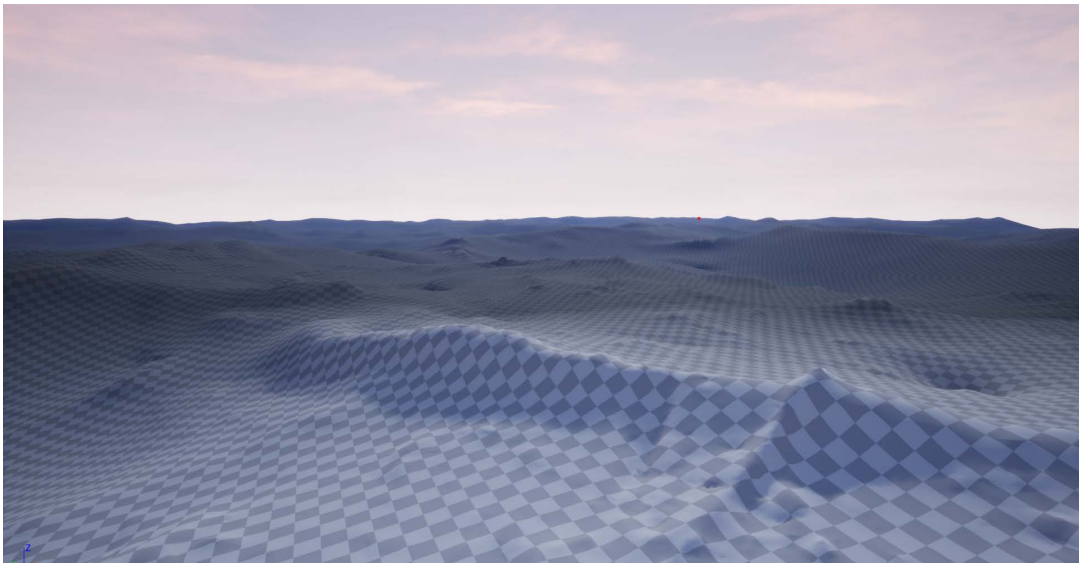

Figure C.2:Screenshot of voxel terrain

This screenshot shows a voxel terrain that was generated using the settings described in this appendix. The terrain is made up of a grid of voxels, each of which is assigned a random value. The values of the voxels are then used to generate the height of the terrain.

Appendix B: CO-PO And CO-PSO Mapping

COURSE OUTCOMES:

After completion of the course the student will be able to

SL. NO	DESCRIPTION	Blooms' Taxonomy Level
CO1	Identify technically and economically feasible problems (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO2	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO3	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions of minimal complexity by using modern tools & advanced programming techniques (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO4	Prepare technical report and deliver presentation (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO5	Apply engineering and management principles to achieve the goal of the project (Cognitive Knowledge Level: Apply)	Level 3: Apply

CO-PO AND CO-PSO MAPPING

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO3
CO1	3	3	3	3		2	2	3	2	2	2	3	2	2	2
CO2	3	3	3	3	3	2		3	2	3	2	3	2	2	2
CO3	3	3	3	3	3	2	2	3	2	2	2	3			2
CO4	2	3	2	2	2			3	3	3	2	3	2	2	2
CO5	3	3	3	2	2	2	2	3	2		2	3	2	2	2

3/2/1: high/medium/low

JUSTIFICATIONS FOR CO-PO MAPPING

MAPPING	LOW/ MEDIUM/ HIGH	JUSTIFICATION
100003/CS622T.1-PO1	**HIGH**	Identify technically and economically feasible problems by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.1-PO2	**HIGH**	Identify technically and economically feasible problems by analysing complex engineering problems reaching substantiated conclusions using first principles of mathematics.
100003/CS622T.1-PO3	**HIGH**	Design solutions for complex engineering problems by identifying technically and economically feasible problems.
100003/CS622T.1-PO4	**HIGH**	Identify technically and economically feasible problems by analysis and interpretation of data.
100003/CS622T.1-PO6	**MEDIUM**	Responsibilities relevant to the professional engineering practice by identifying the problem.
100003/CS622T.1-PO7	**MEDIUM**	Identify technically and economically feasible problems by understanding the impact of the professional engineering solutions.
100003/CS622T.1-PO8	**HIGH**	Apply ethical principles and commit to professional ethics to identify technically and economically feasible problems.
100003/CS622T.1-PO9	**MEDIUM**	Identify technically and economically feasible problems by working as a team.
100003/CS622T.1-PO10	**MEDIUM**	Communicate effectively with the engineering community by identifying technically and economically feasible problems.
100003/CS622T.1-P011	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles by selecting the technically and economically feasible problems.
100003/CS622T.1-PO12	**HIGH**	Identify technically and economically feasible problems for long term learning.
100003/CS622T.1-PSO1	**MEDIUM**	Ability to identify, analyze and design solutions to identify technically and economically feasible problems.
100003/CS622T.1-PSO2	**MEDIUM**	By designing algorithms and applying standard practices in software project development and Identifying technically and economically feasible problems.
100003/CS622T.1-PSO3	**MEDIUM**	Fundamentals of computer science in competitive research can be applied to Identify technically and economically feasible problems.
100003/CS622T.2-PO1	**HIGH**	Identify and survey the relevant by applying the knowledge of mathematics, science, engineering fundamentals.

100003/CS622T.2-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems get familiarized with software development processes.
100003/CS622T.2-PO3	**HIGH**	Design solutions for complex engineering problems and design based on the relevant literature.
100003/CS622T.2-PO4	**HIGH**	Use research-based knowledge including design of experiments based on relevant literature.
100003/CS622T.2-PO5	**HIGH**	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes by using modern tools.
100003/CS622T.2-PO6	**MEDIUM**	Create, select, and apply appropriate techniques, resources, by identifying and surveying the relevant literature.
100003/CS622T.2-PO8	**HIGH**	Apply ethical principles and commit to professional ethics based on the relevant literature.
100003/CS622T.2-PO9	**MEDIUM**	Identify and survey the relevant literature as a team.
100003/CS622T.2-PO10	**HIGH**	Identify and survey the relevant literature for a good communication to the engineering fraternity.
100003/CS622T.2-PO11	**MEDIUM**	Identify and survey the relevant literature to demonstrate knowledge and understanding of engineering and management principles.
100003/CS622T.2-PO12	**HIGH**	Identify and survey the relevant literature for independent and lifelong learning.
100003/CS622T.2-PSO1	**MEDIUM**	Design solutions for complex engineering problems by Identifying and survey the relevant literature.
100003/CS622T.2-PSO2	**MEDIUM**	Identify and survey the relevant literature for acquiring programming efficiency by designing algorithms and applying standard practices.
100003/CS622T.2-PSO3	**MEDIUM**	Identify and survey the relevant literature to apply the fundamentals of computer science in competitive research.
100003/CS622T.3-PO1	**HIGH**	Perform requirement analysis, identify design methodologies by using modern tools & advanced programming techniques and by applying the knowledge of mathematics, science, engineering fundamentals.
100003/CS622T.3-PO2	**HIGH**	Identify, formulate, review research literature for requirement analysis, identify design methodologies and develop adaptable & reusable solutions.

100003/CS622T.3-PO3	**HIGH**	Design solutions for complex engineering problems and perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO4	**HIGH**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.3-PO5	**HIGH**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools.
100003/CS622T.3-PO6	**MEDIUM**	Perform requirement analysis, identify design methodologies and assess societal, health, safety, legal, and cultural issues.
100003/CS622T.3-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts and Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PO8	**HIGH**	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions by applying ethical principles and commit to professional ethics.
100003/CS622T.3-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.3-PO10	**MEDIUM**	Communicate effectively with the engineering community and with society at large to perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering requirement analysis by identifying design methodologies.
100003/CS622T.3-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and prior to that perform requirement analysis, identify design methodologies.
100003/CS622T.4-PO1	**MEDIUM**	Prepare technical report and deliver presentation by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.4-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by preparing technical report and deliver presentation.

100003/CS622T.4-PO3	**MEDIUM**	Prepare Design solutions for complex engineering problems and create technical report and deliver presentation.
100003/CS622T.4-PO4	**MEDIUM**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions and prepare technical report and deliver presentation.
100003/CS622T.4-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and Prepare technical report and deliver presentation.
100003/CS622T.4-PO8	**HIGH**	Prepare technical report and deliver presentation by applying ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
100003/CS622T.4-PO9	**HIGH**	Prepare technical report and deliver presentation effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.4-PO10	**HIGH**	Communicate effectively with the engineering community and with society at large by prepare technical report and deliver presentation.
100003/CS622T.4-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work by prepare technical report and deliver presentation.
100003/CS622T.4-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by prepare technical report and deliver presentation.
100003/CS622T.4-PSO1	**MEDIUM**	Prepare a technical report and deliver presentation to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas.
100003/CS622T.4-PSO2	**MEDIUM**	To acquire programming efficiency by designing algorithms and applying standard practices in software project development and to prepare technical report and deliver presentation.
100003/CS622T.4-PSO3	**MEDIUM**	To apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs by preparing technical report and deliver presentation.
100003/CS622T.5-PO1	**HIGH**	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.5-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by applying engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PO3	**HIGH**	Apply engineering and management principles to achieve the goal of the project and to design solutions for complex engineering problems and design system components or processes that meet the specified needs.
100003/CS622T.5-PO4	**MEDIUM**	Apply engineering and management principles to achieve the goal of the project and use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.5-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO6	**MEDIUM**	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities by applying engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts, and apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO8	**HIGH**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice and to use the engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO1	**MEDIUM**	The ability to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas. Apply engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PSO2	**MEDIUM**	The ability to acquire programming efficiency by designing algorithms and applying standard practices in software project development to deliver quality software products meeting the demands of the industry and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs thereby evolving as an eminent researcher and entrepreneur and apply engineering and management principles to achieve the goal of the project.