Bachelor's Thesis

Information and Communications Technology

2025

Nora Holmberg

Unreal Engine Environment Generation (PCG)

– A Comparative Overview of Existing tools

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2025 | 61 pages

Nora Holmberg

Unreal Engine Environment Generation (PCG)

- A Comparative Overview of Existing Tools

Procedural Content Generation (PCG) is widely utilized in game development to automate the creation of environments, levels, quests, and more. The aim of this thesis was to evaluate the advantages and limitations of Unreal Engine’s newly introduced PCG Plugin, particularly in comparison to existing tools such as the Foliage Tool. The comparison was carried out by creating a Finnish mixed forest using both tools, focusing on how well each supports the creation of realistic game environments. Key points of comparison included usability, customization, control over asset placement, visual realism, and performance. The results were assessed based on the visual quality of the environments and their computational efficiency.

The results showed that the PCG Plugin performs well in large-scale generation and dynamic updates, making it suitable for automated and dense environments. The Foliage Tool, on the other hand, offered greater precision and control, making it better suited for detailed and intentional vegetation placement. Based on the findings, this thesis provides insight for developers, artists, and researchers in selecting tools for realistic environment creation. The optimal tool depends on the needs of the project and the desired balance between automation and manual control.

Keywords:

procedural content generation, unreal engine, game environment design, forest generation, tool comparison, game development.

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2025 | 61 sivua

Nora Holmberg

Unreal Engine -ympäristön luonti (PCG)

- Olemassa olevien työkalujen vertailu

Proseduraalista sisällöntuotantoa (PCG) hyödynnetään laajalti pelinkehityksessä ympäristöjen, tasojen, tehtävien ja muiden elementtien luomisen automatisoimiseksi. Opinnäytetyön tarkoituksena oli arvioida Unreal Enginen uuden PCG-laajennuksen etuja ja rajoituksia erityisesti verrattuna aiemmin käytössä olleisiin työkaluihin, kuten Foliage-työkaluun. Vertailu toteutettiin luomalla suomalainen sekametsä molemmilla työkaluilla, ja siinä keskityttiin muun muassa työkalujen helppokäyttöisyyteen, muokkausmahdollisuuksiin, sijoittelun hallintaan, visuaaliseen realismiin sekä suorituskykyyn. Tuloksia arvioitiin visuaalisen lopputuloksen ja laskentatehokkuuden perusteella.

Työn tuloksena huomattiin, että PCG-laajennus toimii hyvin laajamittaisessa generoinnissa ja dynaamisissa päivityksissä, ja se on sopiva automatisoituihin ja tiheisiin ympäristöihin. Foliage-työkalulla saatiin tarkkuutta ja hallintaa, joten se soveltui paremmin yksityiskohtaiseen ja tarkoitukselliseen kasviston sijoitteluun. Tulosten perusteella saatiin kehittäjille, taiteilijoille ja tutkijoille tietoa realistisiin ympäristöihin tarkoitettujen työkalujen valinnasta. Työkalujen oikea valinta riippuu projektin tarpeista ja halutusta tasapainosta automaation ja manuaalisen hallinnan välillä.

Asiasanat:

proseduraalinen sisällöntuotanto, unreal engine, peliympäristön suunnittelu, metsän generoiminen, työkalujen vertailu, pelinkehitys.

Contents

Figures

Tables

List of abbreviations

Avg.	Average (Own definition)
Culling	The process of not rendering objects that are outside the camera’s view or otherwise unnecessary for performance (Epic Games, n.d.)
GPU	Graphics Processing Unit (Own definition)
Node	A basic unit in a graph-based system representing an operation or function (Epic Games, 2024e)
PCG	Procedural Content Generation (Meegle, 2024)
UI	User Interface (Own definition)

1 Introduction

Procedural Content Generation is a technique widely used in game development that involves using algorithms to generate content automatically. It streamlines the content creation process, allowing for the rapid generation of environments, levels, quests, textures, and more. By utilizing procedural techniques such as fractals, noise functions, and random number generators, PCG enhances the efficiency of game development while improving gameplay through increased variation and replay value.

Unreal Engine, developed by Epic Games, is a powerful real-time 3D creation tool that offers various methods for environmental creation. Among these are the PCG Plugin, which enables algorithm-driven asset placement, and the Foliage Tool, which provides a more manual approach. While procedural tools such as the PCG Plugin facilitate large-scale content generation, manual tools such as the Foliage Tool offer greater precision and artistic control. This study compares these two approaches, evaluating their effectiveness in creating a realistic Finnish mixed forest within Unreal Engine.

Creating realistic environments is crucial for immersive game design. Natural-looking environments enhance believability, helping players feel more engaged and connected to the virtual world. However, achieving realism requires careful attention to asset placement, environmental variation, performance optimization, and an understanding of real-life ecosystems.

A significant challenge in realistic forest generation lies in balancing procedural automation with artistic control. Procedural tools can swiftly generate large environments but may lack the fine detail and natural variation achieved through manual placement by artists. Conversely, manual tools permit precise adjustments but can be labour-intensive and time-consuming for large-scale environments.

To evaluate these tools, a realistic forest environment is created using each tool to assess their functionality and output quality, along with separate

environments for dedicated performance testing. Developing these environments will yield insights into the usability, customization options, and level of control each tool provides over asset placement and the results. The visual outcomes will be evaluated based on the natural appearance and realism of the final product, while performance tests will examine the computational efficiency of each method. The findings are expected to offer valuable insights for game developers, artists, and researchers, assisting them in determining which tool best meets their workflow requirements.

This thesis is structured to provide an in-depth introduction to Unreal Engine and PCG and to explore the significance of realism in game environments. The Tool Overview chapter presents the PCG Plugin and Foliage Tool, followed by the methodology for comparing these two tools regarding their usability, control, and customization. This is followed by a detailed breakdown of the step-by-step process used to create a realistic environmental scene, along with the setup and collection of performance data. The results of the tests are presented and discussed, followed by an analysis of the trade-offs between procedural and manual content generation. Finally, the thesis concludes with key findings and recommendations for future work.

2 About Unreal Engine

Unreal Engine, developed by Epic Games, is a powerful real-time 3D creation engine widely used in video game development across various sizes and genres (Erolin, n.d.; Epic Games, 2025b). It has become one of the most trusted and reliable engines in the gaming industry. Renowned for its exceptional graphical quality, real-time lighting, and advanced rendering capabilities, Unreal Engine has become the preferred choice for AAA studios (Hai, 2022; Epic Games, 2024b), and powers a wide range of popular video games, including Hogwarts Legacy, Final Fantasy VII Remake, Valorant, Fall Guys, and The Stanley Parable. It is also used in titles developed by Epic Games, such as Gears of War and Fortnite (Erolin, n.d.; Wikipedia, 2024).

Beyond gaming, Unreal Engine is extensively utilized in other industries for real-time 3D visualization, virtual production, and simulations (Erolin, n.d.). These applications include virtual production for film and television, architectural visualization, animated content for entertainment or commercials, educational tools, training simulations, and more (Epic Games, 2025b).

2.1 History and Development

Tim Sweeney, the founder of Epic Games, developed the first version of Unreal Engine (Wikipedia, 2025). He began developing the game engine in 1995, intending to use it for a video game project that eventually evolved into the first-person shooter called Unreal. After several years of development, the engine was officially launched with the game's release in 1998 (Wikipedia, 2024).

Initially, Unreal Engine was unavailable to the public and was licensed exclusively to game developers and studios for a fee. In March 2014, this exclusivity changed with the release of Unreal Engine 4, which introduced a subscription-based pricing model at $19 per month (Sweeney, Welcome to Unreal Engine 4, 2014). With the tool now accessible to everyone, the community and content created with it started to expand rapidly.

In 2015, due to the overwhelmingly positive feedback, the subscription model was replaced by a royalty-based system, allowing free access to the engine and all future updates without any upfront costs (Sweeney, If You Love Something, Set It Free, 2015). This decision significantly contributed to the growth and popularity of Unreal Engine's user base, making it more accessible to developers of all sizes, including independent creators and large-scale studios.

2.2 Key features and Capabilities

Unreal Engine offers flexibility for coding through both Blueprints and C++, allowing developers to choose between visual scripting and traditional programming (Epic Games, 2024f). Blueprints are Unreal Engine's visual scripting tool, enabling developers to design gameplay mechanics, animations, UI, and other systems using a simple, node-based interface without the need to write traditional code. This easy-to-use visual approach makes it especially helpful for artists, designers, and those with limited programming experience, allowing them to quickly test, build, and refine game elements (Epic Games, 2024a).

C++ is suitable for more advanced programming. Although it can be challenging to master, it is widely used in game development and is recognized for its excellent memory management, stability, and optimization capabilities (Erolin, n.d.; Epic Games, 2024f; Li, n.d.). It also provides access to deeper levels of the engine, making it the preferred choice for developers seeking more control and performance. The combination of Blueprints and C++ allows Unreal Engine to be adaptable to different project sizes, ranging from indie games to large AAA titles. It facilitates easy deployment across various platforms such as iOS, Android, Windows, PlayStation, and Xbox (Erolin, n.d.; Hai, 2022).

3 About Procedural Content Generation

Procedural Content Generation is the automated generation of content using algorithms instead of manual creation (Risi & Togelius, 2019; Meegle, 2024). While game developers primarily utilize PCG in game development, it is also applied in simulations and animated movies (Van Brummelen & Chen, n.d.). By implementing a set of predefined rules or procedures, PCG automates the content generation process, allowing for the creation of large, complex environments and assets without direct human intervention.

One of the main advantages of PCG is its efficiency. By automating content generation, developers can significantly reduce production time and costs, which is especially valuable for large-scale open-world games or simulations that require generating vast amounts of content quickly and consistently. In addition to efficiency, PCG introduces diversity and randomization into the generated content. Procedural algorithms, such as fractals, noise functions, and random number generators, ensure that the content varies with each execution, adding further complexity and richness to the experience (Game-Ace, 2024a). These benefits enhance versatility and scalability and improve the gaming experience by increasing the game's replay value, allowing players to encounter new scenarios each time they play (Risi & Togelius, 2019; Game-Ace, 2024a). However, the success of PCG depends on the quality of the algorithms used and the data input. Poorly designed rules can result in repetitive or unnatural content that lacks the desired immersion or believability.

3.1 Applications in game development

PCG has become an essential tool in game development, particularly for large-scale, open-world games and simulations. It is utilized across various stages of development, contributing to both the design phase and runtime processes.

Developers often use PCG for level game design, including creating landscapes, terrains, and textures (Aversa, 2015; Van Brummelen & Chen, n.d.; Game-Ace, 2024a; Smith). It also assists in populating these environments by generating various assets, such as trees, structures, and other environmental elements, which help populate game worlds with minimal manual effort.

At runtime, developers frequently apply PCG to create diverse and dynamic gameplay experiences that enhance the immersion and replay value of the game. For instance, it generates randomized levels and dungeons, dynamic quests and loot, and events like random encounters or in-game disasters (Aversa, 2015). Additionally, PCG can sometimes simulate environmental factors, such as weather systems, further enhancing the game's dynamic and immersive qualities.

A well-known example of PCG, particularly in terrain generation, is the creation of planets in the game No Man's Sky, where computers algorithmically generate 18 quintillion (18, 000, 000, 000, 000, 000, 000) unique planets and moons that players can explore (Van Brummelen & Chen, n.d.).

3.2 Common pitfalls and solutions

Procedural generation has many benefits in game development, but it also brings challenges that developers must carefully manage. One common issue is the risk of generating repetitive or uninteresting content, which can make game worlds monotonous. To avoid this, developers must incorporate sufficient variability within their algorithms and strategically blend procedural elements with handcrafted designs to maintain diversity (Game-Ace, 2024a). Another challenge is the unpredictability of procedural generation. Because developers dynamically create content, ensuring it aligns with the intended design can be challenging. Fine-tuning algorithms to achieve specific results often requires extensive adjustments, making it difficult to maintain complete creative control (Meegle, 2024). This unpredictability can be especially problematic in games that rely on structured-level design or narrative consistency. Beyond creative

concerns, performance is a key consideration. Complex procedural generation algorithms can demand significant computational resources, leading to performance bottlenecks if improperly optimized. These challenges are particularly evident in real-time applications, where the game generates content dynamically during gameplay. Without careful optimization, procedural systems can impact frame rates and overall game responsiveness, making performance management a crucial aspect of PCG implementation (Latif, Zuhairi, & Qudus Khan, 2022). By addressing these challenges, developers can better harness the strengths of procedural generation while minimizing its drawbacks.

4 Realism in Games

4.1 Importance of Realism in Game Design

When working on landscapes, creating realistic environments significantly enhances the gaming experience. Realistic landscapes contribute to the overall atmosphere and mood of the game (Toolfarm, 2024), and enhances immersion, allowing players to feel as though they are genuinely part of the game world. This sense of realism is often referred to as "spatial presence", where players perceive the game environment as real (Wirtz, 2023; Madigan, 2010).

When creating a realistic landscape, it is essential to reference the real world, as realism largely depends on how natural the environment appears. Observing details from actual landscapes helps in understanding key aspects of creating digital landscapes, including natural growth patterns, proportions, lighting, ground cover, topography, and biome-specific plant types and foliage.

4.2 What Contributes to a Realistic Landscape

Creating a realistic landscape requires more than just selecting visually appealing assets. Several key factors must be considered, including the choice of assets, their placement and size, graphics and performance, and the environment's overall design.

Assets

The visual quality of assets plays a significant role in crafting a believable environment. Key aspects include realistic textures and details, accurate scale, proportion, and colour for foliage. Well-designed assets should seamlessly blend into their surroundings, reinforcing the overall atmosphere and immersion of the environment (Karppinen, 2023).

While biome-specific assets are not strictly necessary, they considerably enhance realism. By considering the biome, such as forests, deserts, swamps, or tundra, or environmental factors like moisture levels, sunlight, and temperature, the appropriate foliage can be selected even for fantasy-themed environments, helping to create grounded and immersive landscapes.

Asset Placement

Emulating natural growth patterns is essential for realism. In nature, trees and plants do not grow in uniform rows, perfect geometric patterns, or overlap. Clipping or floating assets disrupt the illusion of a natural environment, creating a sense of disconnection for the player. Unnaturally placed assets diminish the immersive quality of the world, disrupting the player's connection to the environment.

Graphics and Performance

While realism is key, developers must also balance it with performance. High-quality graphics, including textures, lighting, and shadows, significantly contribute to a lifelike environment and immersion (Game-Ace, 2024b). However, it is vital to optimize these assets to prevent overwhelming the system, especially when working with large-scale landscapes. Careful consideration of asset placement and level of detail is necessary to achieve the right balance between visual fidelity and performance.

Diversity

Nature does not repeat itself perfectly (Toolfarm, 2024), and developers must reflect this diversity in digital landscapes. A monotonous or repetitive environment can feel artificial and static. Randomizing the size, rotation, and placement of assets can help break up repetition and create the illusion of

diversity without requiring an excessive number of different models. Additionally, minor environmental details, such as subtle foliage movement, wildlife behaviour, and ambient sounds, can further enhance realism and immersion (Black, n.d.). The PCG Plugin and the Foliage Tool provide features that facilitate this level of customization, making it easier to introduce variability while maintaining performance.

5 Tool Overview

5.1 Foliage Tool Introduction

The Foliage Edit tool is a well-established feature in Unreal Engine that allows developers to efficiently populate vast open-world environments with vegetation and static meshes through a brush-like interface. This tool enhances both the visual richness and the immersive quality of game worlds in a short time. It offers two main placement methods: the Paint Tool, which allows for precise manual placement, and the Fill tool, which enables users to apply assets over large areas quickly. These tools come with options for density adjustments, random scaling, rotation variation, and alignment control, allowing developers to customize the appearance of their assets in a scene. These features allow for greater flexibility and customization, providing developers with a high degree of control over the placement and variation of assets in each scene, making it possible to achieve a diverse and dynamic environment (Epic Games, 2024c).

Epic Games created this tool to address limitations in earlier systems, such as the one in Unreal Engine 3, which experienced performance issues and inadequate interaction with Unreal Lightmass. By utilizing advanced techniques for efficient asset rendering, the new Foliage tool handles large batches of assets more effectively, reducing draw calls and improving frame rates. The refined system, available since the launch of Unreal Engine 4 in 2014, has contributed to more optimized workflows in large-scale world creation (Epic Games, 2011b). Early versions were released during the Unreal Development Kit Beta in 2011 (Epic Games, 2011a).

5.2 PCG Plugin Introduction

The Procedural Content Generation framework plugin is a recent addition to Unreal Engine. It was showcased at the Game Developers Conference in May 2023 as an experimental feature with the launch of Unreal Engine 5.2. This feature is still under development, and more updates are expected in the future (Epic Games, 2023).

The PCG plugin is a powerful and flexible toolset designed to enable technical artists, designers, and programmers to create dynamic, procedural content and tools of any complexity within the Unreal Engine (Epic Games, 2024d).

The PCG Plugin utilizes Unreal Engine's visual scripting system, Blueprints, as its primary coding language, providing a graph-based framework for procedural asset generation. Users can easily adjust key parameters, such as randomness, distribution patterns, rules, and environmental variables, through the intuitive node-based interface, with C++ available for more advanced control (Epic Games, 2024e). It features in-editor tools and a runtime component, allowing users to establish rules for asset placement and dynamically generate and modify environments in real time. This setup enables creators to fine-tune content generation to fit their specific project needs while optimizing performance by allowing the environment to adapt based on runtime parameters (Epic Games, 2023). Due to its versatility, the PCG plugin can handle various tasks, from adding small environmental details like moss or mushroom growth on surfaces to large-scale environments such as buildings, forests, biomes, and even entire worlds or planets (Epic Games, 2024e; DiLaura, 2023). This flexibility empowers developers to craft immersive and responsive gameplay experiences (Epic Games, 2023).

6 Method of Comparison

This section outlines the methodology used to compare Unreal Engine's PCG Plugin and the Foliage Tool. The comparison evaluates their effectiveness in generating realistic forest environments, with the aim of identifying each tool's strengths and limitations in terms of workflow efficiency, visual output, and system performance.

A Finnish mixed forest was selected as the case study for several reasons. First, there is limited research or practical use of Finnish nature as a reference for environment generation in game development, making it a unique and culturally relevant choice. Second, the biome offers a balanced variety of tree species and foliage, enabling the creation of a realistic yet manageable scene without the complexity of more diverse ecosystems.

6.1 Tool Selection

The PCG Plugin, a recently introduced tool, was selected to assess its advantages and limitations in comparison with the well-established Foliage Tool. Both tools were chosen due to their relevance and different approaches in forest and vegetation generation within Unreal Engine. The PCG Plugin represents a procedural, rule-based approach, while the Foliage Tool offers a manual, artist-driven workflow. This comparative study aims to highlight the strengths and weaknesses inherent in each approach, thereby offering a clearer understanding of the trade-offs between procedural automation and manual control in environmental generation.

6.2 Comparative Framework

The evaluation is structured around key criteria that influence tool selection and effectiveness in game development:

Usability and Workflow:
Assesses accessibility, learning curve, and overall efficiency.

Customization and Control: Evaluates the degree of control users have over asset placement, distribution, and settings.

Documentation and Support: Reviews the availability and quality of official documentation, tutorials, and community guidance.

Realism and Output Quality: Examines the natural appearance and visual accuracy of the generated environments.

Performance and Optimization: Analyses computational efficiency and system resource usage under various conditions.

These criteria encompass both the technical and practical aspects of forest generation, ensuring a well-rounded evaluation of both tools.

6.3 Data Collection and Sources

The comparison was informed by both primary and secondary data sources:

Primary Data: Hands-on testing will involve creating Finnish mixed forests using both tools. The same asset packs and environmental parameters were applied to ensure consistency. This direct comparison will evaluate workflow efficiency, customization capabilities, and output quality under controlled conditions. Additional performance testing was conducted in controlled test environments to measure system resource usage and runtime behaviour.

Secondary Data: Additional insights will come from online discussions, forums, and official documentation. User experiences will be reviewed from multiple sources to provide a broader perspective on the strengths and limitations of each tool. For fact-based data, only official Epic Games documentation, relevant academic literature, and other credible sources were consulted to ensure the accuracy, reliability, and comprehensiveness of the information.

6.4 Reliability and Validity

To ensure a fair comparison, both tools were tested under identical conditions, including the same terrain, assets, and density levels. Performance tests were executed using Unreal Engine's built-in CSV profiling tool, with multiple repetitions conducted to ensure consistency. Secondary sources were cross-referenced to confirm the validity of supporting information.

This methodology aims to produce replicable, well-rounded, and practically useful insights for developers seeking the appropriate tool for realistic forest generation in Unreal Engine.

7 User Experience

7.1 Usability and Workflow

Foliage Tool

Since the Foliage Tool is a built-in feature of Unreal Engine, it integrates seamlessly with the engine's systems. Directly accessible from the standard toolbar, the tool offers a clean interface, as illustrated in Figure 1, where the most important settings, such as tree density and collision parameters, are clearly displayed in the left panel (Epic Games, 2024c).

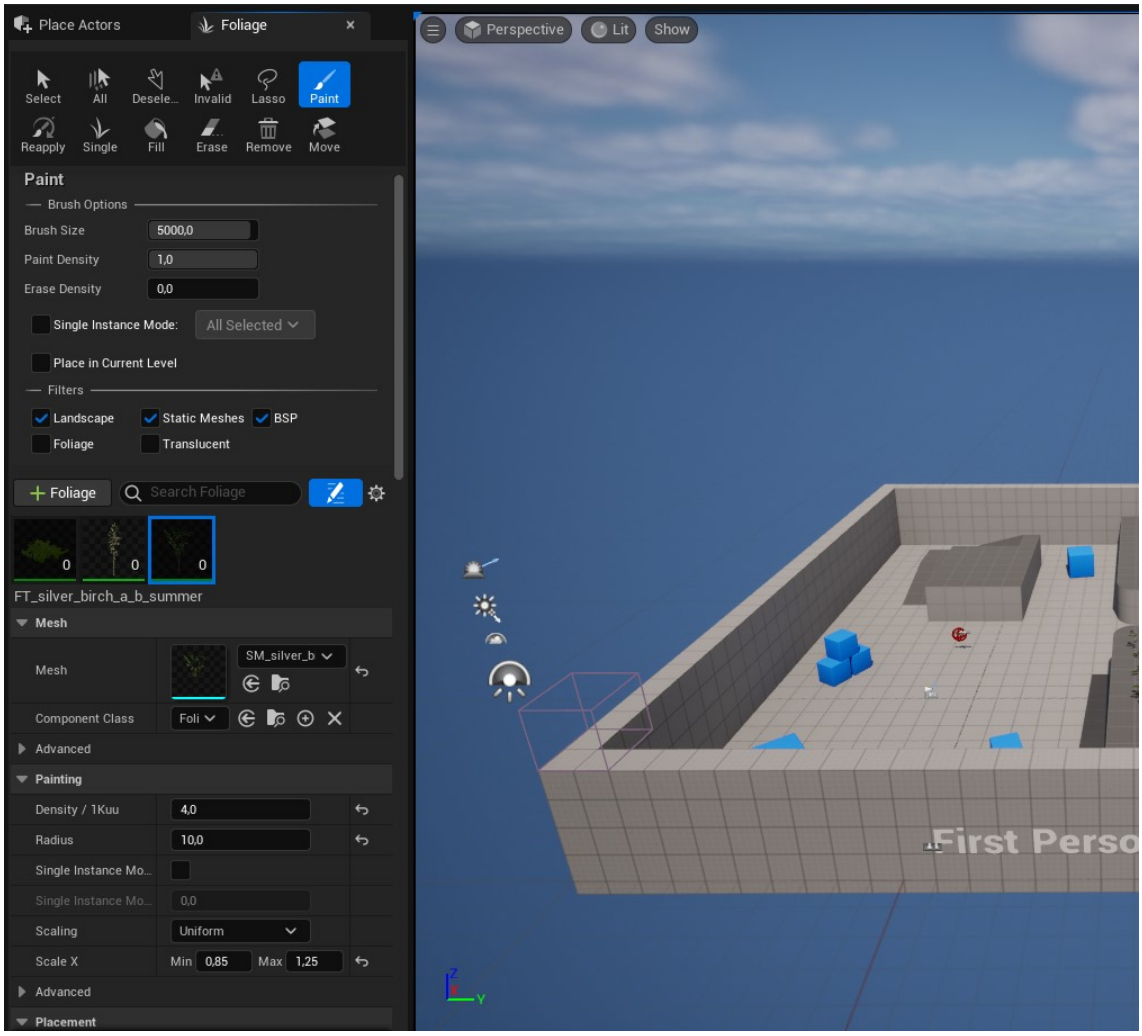

Figure 1. Screenshot of the Foliage Tools interface in Unreal Engine 5.

The interface is intuitive, with all controls available without needing to navigate additional menus. It’s simple, brush-based workflow allows users to efficiently select, paint, and erase foliage assets, offering a flexible and streamlined approach to asset placement. To get started, users can choose and toggle the desired foliage assets in the Foliage Palette, adjust the necessary settings, and apply them directly to the scene (Epic Games, 2021).

The tool emphasizes user-defined areas for foliage placement, typically utilizing terrain or mesh surfaces as the foundation. It allows users to toggle between different surfaces, such as Landscapes, Static Meshes, and Foliage, which enhances precision and improves workflow efficiency (Epic Games, 2024c). Additionally, when users move a mesh or terrain, its foliage automatically moves with the surface, reducing the need for manual adjustments and making asset repositioning faster and more intuitive.

While it does not offer fully automated asset generation, the tool's flexibility enables artists to create diverse, customized scenes without requiring extensive manual intervention. Although users can only adjust asset and tool settings, its simplicity ensures a smooth workflow, making it accessible and easy to use. With a low learning curve, it suits users across all experience levels, from beginners to experts.

PCG Plugin

The PCG Plugin is seamlessly integrated into Unreal Engine's editor, providing an intuitive approach to procedural content generation. Its node-based interface offers a visual and accessible workflow, allowing users to easily create and modify procedural environments. By selecting, dragging, and connecting nodes within a PCG graph, users can define rules for asset placement, density, transformations, and other environmental variables (Epic Games, 2024e). When a node is clicked, all its available settings are displayed to the user. Figure 2 provides a visual representation of a blank PCG graph, offering a glimpse of the interface's layout and structure, along with the available node categories for building procedural content generation workflows (Epic Games, 2024d).

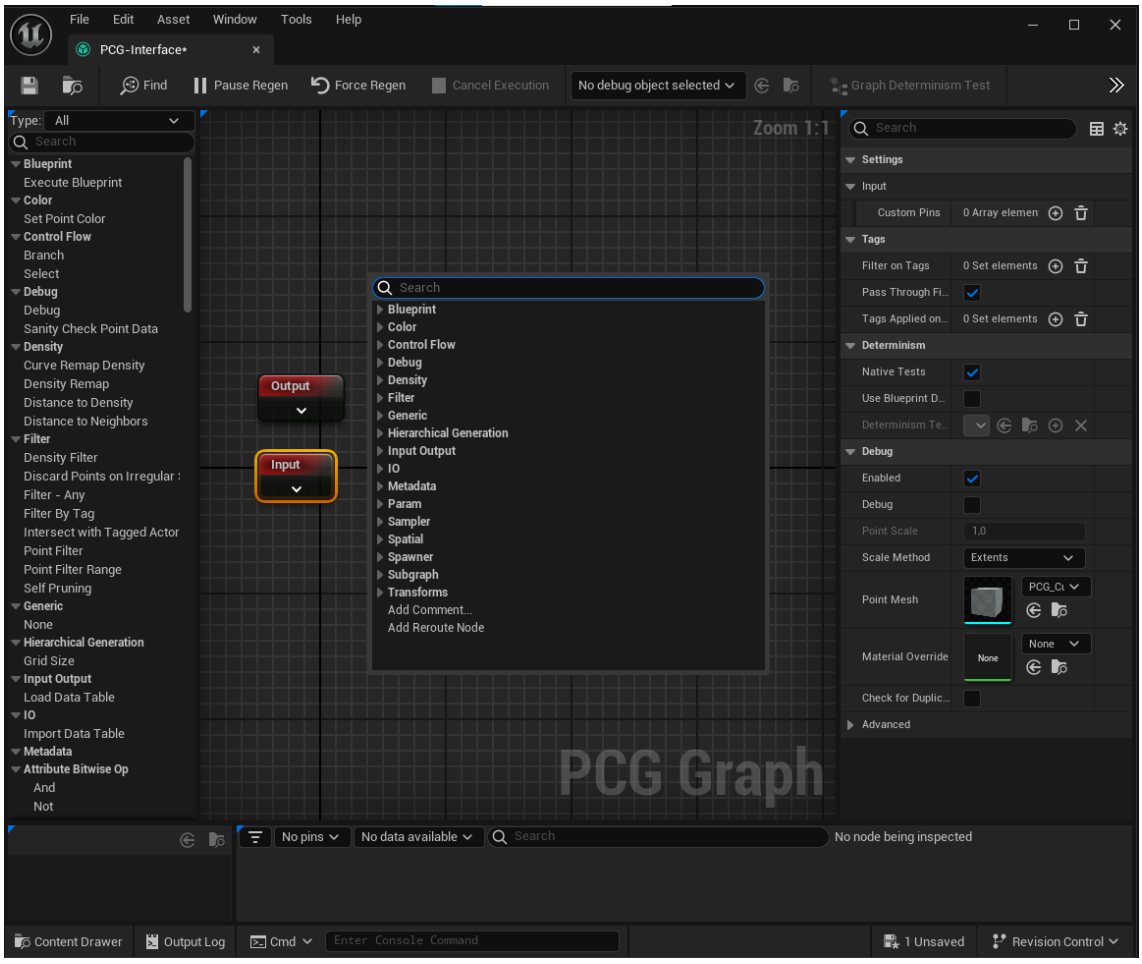

Figure 2. A screenshot of a blank PCG graph in Unreal Engine 5, showcasing the node-based interface for procedural content generation.

A PCG graph requires three essential nodes to function: Input, Surface Sampler, and Static Mesh Spawner (Deacon, 2024). The Input node gathers surface data, the Surface Sampler determines placement and density, and the Spawner generates and places assets accordingly. Each graph links to a PCG Volume, which defines the area where procedural content appears. This volume can be positioned, adjusted, and reused across multiple locations within a scene (Epic Games, 2024e).

PCG graphs are highly reusable and easily modifiable, making them valuable tools for iterative development. Developers can efficiently replicate procedural setups in different areas, ensuring consistent asset placement while saving time. The PCG UI also includes built-in visual debugging tools, making troubleshooting user-friendly (Epic Games, 2024e). These tools help users quickly identify issues and refine their procedural generation logic.

However, while the PCG workflow aims to be accessible, understanding how different nodes interact requires some learning, which may not be immediately intuitive for beginners, resulting in a steeper learning curve.

7.2 Customisation and Control

Foliage Tool

The Foliage Tool provides various customization and control options, enabling users to place and modify vegetation assets in their environments efficiently. Its brush-based interface offers flexibility for manual placement, granting users precise control over individual elements while maintaining efficiency when covering extensive areas.

The Paint and Fill tools address different customization needs. The paint tool allows for accurate, manual placement with adjustable settings such as brush size and asset density, while the Fill tool facilitates rapid application over larger surfaces, streamlining the process of populating environments quickly. Each asset has adjustable settings that users can manage through sliders and checkboxes (Epic Games, 2024c). Commonly adjusted settings include Placement Settings, like rotation and size. The 'Collision with World' setting prevents overlap with other objects, while the 'Radius' setting adjusts the spacing between assets. The 'Density' setting determines how many instances are placed within a specified area, assisting in controlling the distribution of assets. The tool offers additional control over asset placement, providing options like placement restrictions based on slope angle or specific landscape layers.

In the Foliage Tool, placed assets retain their initial settings, which means any changes made within the workflow will not automatically impact already-placed assets. It is important to note that this limitation restricts the ability to modify certain properties of the assets after placement. To modify existing assets, users can utilize the Select Tool or the Reapply Tool, depending on the specific adjustments they wish to make. The Select Tool is perfect for fine-tuning individual assets, allowing users to adjust their rotation, position, and scale or remove unwanted elements to create a more refined environment.
For more extensive modifications, the Reapply Tool enables users to select

specific assets from their asset list, choose which settings to adjust, and then paint over the placed assets to apply the updated values (Epic Games, 2021). However, it is crucial to note that not all settings can be modified. For instance, users cannot adjust settings like culling on already placed assets, which can significantly limit the fine-tuning of asset distribution in large-scale environments.

With its blend of manual precision and large-scale placement tools, the Foliage Tool excels in detailed asset placement. It is a powerful and versatile option for creating tailored, realistic environments.

PCG Plugin

The PCG Plugin offers extensive customization through its node-based scripting system. The PCG Graph provides a range of nodes for filtering and controlling asset placement, including the ability to spawn actors. Each node performs a specific procedural action or process, such as filtering by proximity or terrain, and includes settings and parameters for further fine-tuning. Other standard node-based operations involve scattering assets randomly across a surface, aligning objects to terrain curvature, and ensuring assets do not overlap. In addition to the built-in nodes, developers can create custom nodes to extend procedural workflows by integrating unique functionality not available in the default set using C++ or Blueprints (Logut, 2024). Custom nodes allow users to define specific operations, such as manipulating position or colour, for enhanced control over procedural generation. These nodes seamlessly integrate into the node graph, preserving an intuitive workflow and enabling flexible, reusable content creation across projects. Achieving specific outcomes, such as detecting and spawning assets on flat surfaces, often requires combining multiple nodes. For instance, when combined with the Density filter, the Normal to Density node enables artists to filter out points located on slopes. By organizing nodes in this manner, users can achieve various outcomes, further enhancing control over asset placement and enabling

more precise environmental designs (Deacon, 2024). Constraints can also be applied to procedural placement, such as setting minimum and maximum asset densities, ensuring that objects always appear in specific locations, or preventing assets from being positioned too closely to existing geometry. These constraints help maintain oversight over the procedural system while permitting variation.

A standout feature of the PCG Plugin is its runtime component, which facilitates real-time updates to the environment whenever any changes have been made (Epic Games, 2024e). This dynamic feedback significantly boosts the efficiency of the design process, allowing for quicker iteration, testing, and fine-tuning. As the system relies on defined rules instead of manual placement, the immediate visual feedback empowers users to experiment with different parameters and constraints, swiftly refining the generated assets to fit the intended environment better. The PCG UI also includes built-in debugging tools that assist in visualizing how nodes interact with the scene and with one another. The visual debugger highlights potential issues or areas for improvement, making it easier to troubleshoot and refine procedural generation logic (Epic Games, 2024e). This graph-based approach is ideal for users looking to automate content creation while maintaining control over essential aspects such as asset placement and environmental variation. By leveraging the flexibility of nodes, real-time feedback, and debugging tools, artists and developers can fine-tune procedural generation to produce detailed and optimized game environments.

7.3 Documentation and Support

Official Text Documentation

Clear and detailed official documentation is available for the Foliage Tool and PCG Plugin. These pages cover essential aspects such as setup instructions, configuration settings, and workflows. The Foliage Tool documentation emphasizes its features and workflows (Epic Games, 2024c), while the PCG Plugin documentation outlines the PCG framework, including graph creation, attributes, and parameters, along with a step-by-step tutorial for getting started with the basics (Epic Games, 2024e). The PCG Plugin documentation also includes an introductory tutorial series by Adrien Lout, a Tool Programmer at Epic Games (Logut, 2024). This series assists new users by offering short, topic-focused videos that help them find relevant information quickly. All official documentation is accessible on Epic Games' official Unreal Engine website (Epic Games, 2025a), which features forums for user discussions and learning materials created by both Unreal Engine and the community.

Official Video Documentation

Unreal Engine maintains a verified YouTube account (Unreal Engine, 2011) and regularly publishing official video documentation for its tools. The latest official video tutorial for the Foliage Tool was released in 2017 (Unreal Engine, 2017). While still relevant due to the consistency of core features, these videos may not cover the latest updates or new functionalities introduced in recent versions of Unreal Engine. The PCG Plugin benefits from regular video updates, with new videos typically released every few months. These videos focus on introducing new features and updates and providing in-depth tutorials for effectively using the plugin.

Community Documentation

In addition to official documentation, the Foliage Tool and the PCG Plugin benefit from a wide range of community-created resources, including tutorials, guides, and discussions on platforms such as YouTube, Reddit, Art Station, and Discord. These community tutorials address various aspects, from basic setup to advanced techniques, offering additional learning opportunities beyond official materials. While the quality and accuracy of community tutorials can vary, they may not always reflect the latest updates to Unreal Engine. These resources can be valuable for troubleshooting and learning new techniques, but users should verify information against official documentation to ensure reliability.

8 Realism and Output

This section compares the PCG Plugin and the Foliage Tool regarding their ability to create realistic forest environments, specifically focusing on a Finnish mixed forest. It evaluates the user-friendliness and efficiency of each tool by examining factors such as workflow challenges, the time and effort required, and the quality of the results. The analysis highlights the strengths and limitations of both tools, providing valuable insights into asset distribution patterns, customization options, and overall output quality.

8.1 Observations and Key-Features

The focus is on recreating the observed natural distribution of trees and foliage to create a realistic landscape utilizing both tools. Reference photos of Finnish mixed forests, such as Figure 3, were a primary guide to identifying and replicating the key characteristics, including tree composition and terrain features. These forests are predominantly composed of Scots Pine (67%), Norway Spruce (23%), and Birch (10%), as confirmed by both sources and photographic references (Valtsa, 2020). The terrain is relatively flat, interspersed with occasional small hills, and primarily covered with bilberry bushes, sparse ferns, and underbrush. Scots Pines, spaced approximately 2 to 6 meters apart, contribute to an open, spacious atmosphere, allowing visibility between trees, while scattered young spruce and birch saplings enhance diversity in the undergrowth. These observations inform specific settings in the Foliage Tool and PCG workflows, such as density settings, distribution constraints, and alignment rules. With these considerations in mind, the landscape creation process can commence.

Figure 3. Reference photo of a Finnish mixed forest, highlighting typical tree composition and terrain features. Part of a larger set used for asset selection and forest creation.

8.2 Asset Selection and Optimisation

Assets from the Nordic Auto Biome and Temperate Vegetation: Spruce Forest were utilized in the PCG and Foliage Tool workflows to recreate the environment accurately (Realpixels Studio, 2024; Project Nature, 2024). These assets effectively represent Nordic vegetation, reflect the composition of a typical Finnish mixed forest, and provide high-quality models optimized for game environments, as illustrated is Figure 4.

Nature does not repeat itself perfectly (Toolfarm, 2024), and while imperfections and variations enhance realism, they can impact performance. To address these performance challenges, focusing on optimization planning is critical to balance visual fidelity with performance and prevent gameplay disruptions caused by lag or frame drops. To balance performance and variety, only 10

unique assets were selected, including variations of Scots Pine, Spruce, and Birch trees, shrubs, bilberry bushes, and stones, thus reducing memory overhead and preserving visual diversity. For optimization, minor details like rocks and shrubs were assigned short culling distances to enhance performance, while assigning longer culling distances to primary tree models, such as the Scots Pine and Spruce, to uphold the appearance of depth and visual appeal of the forest.

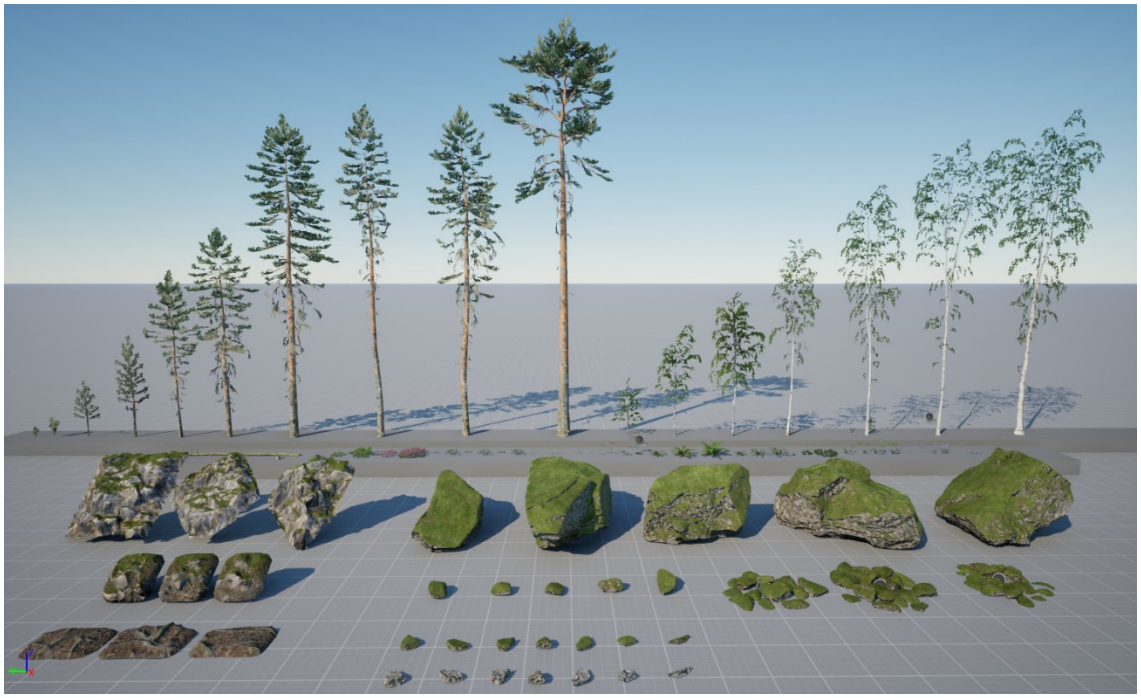

Figure 4. Screenshot from Unreal Engine of all the assets available in the Nordic Auto Biome asset pack.

8.3 Creation Process

Variation was introduced with both tools to achieve a realistic depiction of a Finnish mixed forest by adjusting the scale and rotation parameters for all assets. These adjustments ensured that no two instances of the same asset appeared identical, contributing to the natural randomness observed in real forests. Consistency was ensured using the same assets in both environments and similar settings in both tools.

Tool Setup

To set up the Foliage Tool, the desired assets were added to the asset list, and each asset's settings were adjusted. The Placement Settings, including rotation and scale, were crucial in avoiding repetition. Collision with World prevented assets from overlapping with other objects, while the Radius setting defined the distance between them, and density defined the number of instances. For larger trees like Pine and Spruce, density was kept lower to avoid overcrowding, while for ground cover assets like Bilberry bushes, higher density was used to create a more detailed forest floor.

Placement Strategy

When building the forest using the Foliage Tool, the process began with testing asset placement using a smaller brush size. This approach allowed for fine-tuning the settings while minimizing performance impact by placing fewer assets during testing. The plan was to switch to a larger brush size once a realistic placement was achieved, enabling efficient landscape coverage with one brush click. However, a challenge arose when increasing the brush size. Since the brush size and density settings are linked, increasing the brush size affected the asset density inadvertently, resulting in a more spread-out distribution than intended. As a result, replicating the initial asset spacing and coverage achieved during testing became difficult. Testing asset placement with a brush size that closely matches the one intended for the final scene could help avoid this issue.

Ground Coverage

To replicate the uneven and textured forest floor observed in the reference photos, bilberry bushes were placed with random vertical placement in the scene, introducing height variation and enhancing the sense of natural irregularity. In the Foliage Tool, this was achieved by modifying the "Z Offset"

setting, allowing for the definition of a range for the bushes' spawning height. Similarly, in the PCG Plugin, the vertical placement was adjusted by modifying the Transform node's "Z value" range. A moss texture was applied to the forest floor to enhance the terrain's authenticity, providing visual continuity and blending seamlessly with the bilberry bushes. Together, these elements contributed to a realistic depiction of the bumpy, irregular forest floor characteristic of Finnish landscapes.

Tree Distribution

Once the forest floor was established, the next step focused on tree placement. Observations from the reference photos informed the density and arrangement of tree species. Since spruce trees are less abundant but contribute significantly to the forest's background coverage, their correct density was prioritised. Given the large size of the spruce trees, it was essential to prevent overlaps with other assets to maintain a natural appearance.

In the Foliage Tool, this issue was initially addressed by adjusting the "Collision with World" setting for the spruce trees, ensuring that no assets spawned within a specified distance of them. However, challenges arose when trying to paint both pine and spruce trees together, as the dense spawning of pine occupied most of the available space, leaving insufficient room for spruce to spawn. As a workaround, the density of spruce trees increased significantly. However, this solution proved unsustainable in the long term, as it did not resolve the underlying collision and prioritization issues. The PCG Plugin managed the overlap problem using the 'Bounds Modifier' node to define specific boundaries around the spruce trees. The 'Difference' node was then applied to filter out any assets that overlapped with these boundaries, ensuring proper spacing. These adjustments, as shown in Figure 5, allowed for a more consistent and natural tree distribution, resulting in a realistic forest layout.

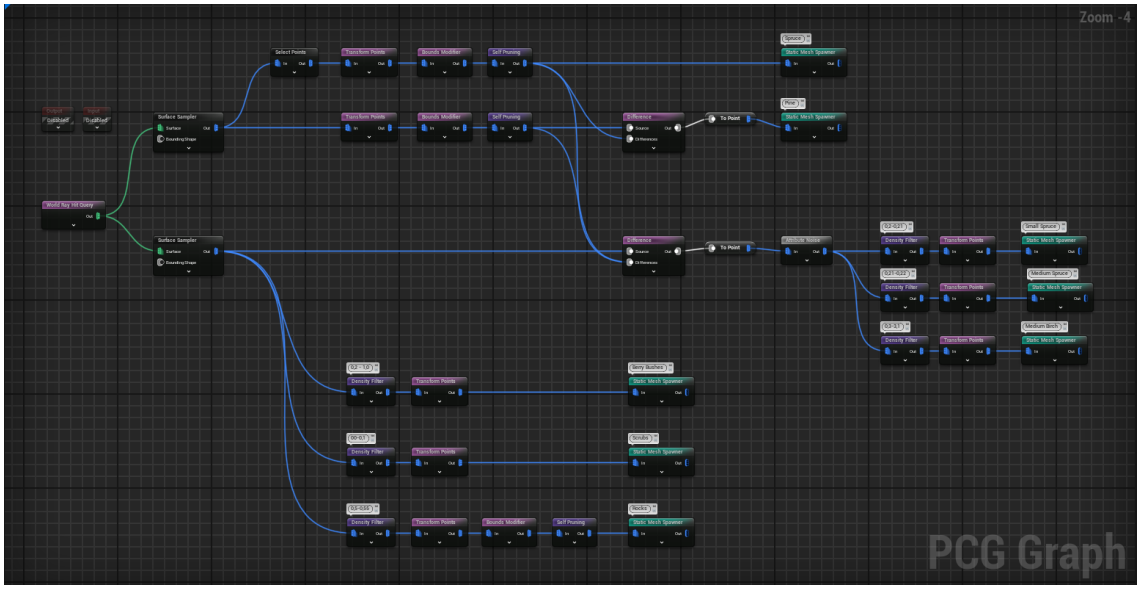

Figure 5. Screenshot of the PCG Graph used to generate a realistic forest landscape with the PCG plugin.

Detailing

With the primary tree placement completed, the final stage focused on detailing the undergrowth to enhance the forest's realism. Reference photos revealed the presence of young spruce trees scattered throughout the landscape, contributing to the forest's natural density and visual complexity. To replicate this, two distinct small spruce tree assets were used, providing variation in both colour and appearance. Additionally, various shrubs and rocks were scattered throughout the scene to break uniformity and introduce organic imperfections. While subtle and easy to apply using both tools, these details were crucial in grounding the scene and contributing to the forest's overall realism.

8.4 Results

This section presents the results of testing the Foliage Tool and PCG Plugin in generating a realistic Finnish mixed forest. The focus is on evaluating their performance in terms of workflow, usability, realism, and output, with comparisons made to reference photos.

The Foliage Tool and PCG Plugin produced realistic Finnish mixed forests that closely resembled the reference photos in overall structure. The placement of trees, shrubs, and other elements followed the natural patterns observed in the reference images. However, the reference photos depicted lusher environments with a greater diversity of plant life, while the generated forests were intentionally limited in asset variety to prioritize performance. As a result, the generated environments appeared slightly less dense and varied, as shown in Figure 6 and Figure 7.

Figure 6. Forest Generated using the PCG Plugin.

Additionally, the Foliage Tool encountered challenges in asset placement along terrain transitions, leading to strips of exposed ground that occasionally disrupted the seamless integration of the forest.

Figure 7. Forest Created using the Foliage Tool.

The two tools demonstrated noticeable differences in their asset distribution methods. The Foliage Tool uses a randomized distribution method, placing assets in an organic yet chaotic manner. This randomness enhances the natural feel of the environment, mimicking the irregular nature of ecosystems. However, this randomness also has potential drawbacks. Assets may become tightly clustered or overly sparse, leading to undesirable gaps, overcrowding, and asset overlapping. These issues were resolved by adjusting settings like density, radius, and collision with the world, but they were not entirely avoided.

In contrast, the PCG Plugin offers a structured and algorithmically controlled approach, allowing for even spacing and predictable placement. However, this method can sometimes lack the natural irregularity found in real ecosystems, occasionally resulting in a grid-like pattern. To address this issue, performing additional fine-tuning of randomness parameters was necessary to achieve a more natural distribution.

Figure 8 illustrates a visual comparison of the two distinct asset placement methods, highlighting the differences in distribution styles between the tools.

Figure 8. Comparison of asset distribution. Foliage Tool (randomized) vs. PCG Plugin (structured).

This visual representation helps to better understand the contrast between the Foliage Tool's organic randomness and the PCG Plugin's structured approach.

9 Performance

This section compares the performance of the PCG Plugin and the Foliage Tool in terms of their efficiency in generating large-scale forest environments. It evaluates key performance factors such as frame rate, memory usage, and thread utilization, while testing various levels of forest density and camera angles. The analysis highlighting each tool's scalability, optimization, and resource management. The findings provide valuable insights into the performance capabilities of both tools under varying conditions and offer a clear understanding of their suitability for large, complex scenes.

9.1 Test Setup

Test Environment Setup

The Fill Tool was used with the Foliage Tool instead of the Paint Tool during the creation of the test environment because it provides better control over asset placement and ensures consistent asset counts. However, due to performance constraints, the Fill Tool cannot spawn assets directly on landscapes; thus, testing for both tools was conducted on a flat surface mesh, which served as a simplified surface to ensure uniform conditions

In the PCG Plugin, the test environment was created using a simple procedural generation graph. As depicted in Figure 9, this graph utilizes only the essential nodes discussed previously in the Usability and Workflow sections. These nodes include the Input node, which, in this case, is the World Ray Hit Query, the Surface Sampler for surface information, the Transform node for positioning, and the Static Mesh Spawner for placing the tree assets. This configuration facilitated efficient procedural generation while keeping the environment simple and focused on performance testing.

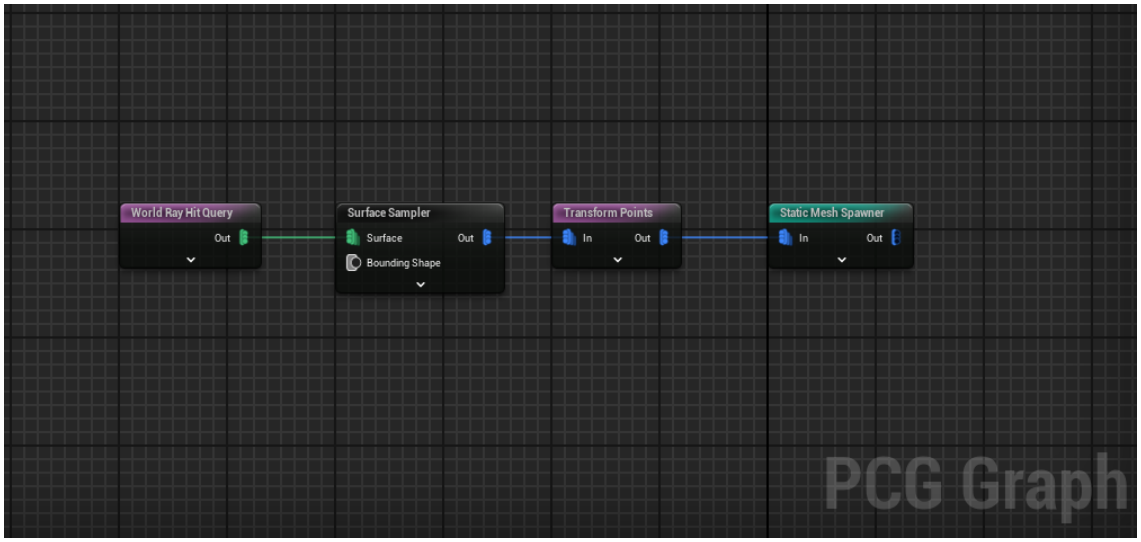

Figure 9. Setup of the PCG Graph in Testing Performance.

Performance Testing Conditions

Results were collected from three separate test batches conducted on different dates. Performance tests were conducted using three levels of forest density: low, medium, and high. Each environment was tested across four camera angles: corner-to-corner, centre, rotating in the centre, and moving. Each camera angle was tested five times per density level to ensure consistency, with each test lasting 30 seconds. Testing was performed sequentially from low to high density, with Unreal Engine restarted between camera angles, and the PC rebooted when switching between the PCG Plugin and the Foliage Tool.

Testing various densities and angles is crucial for understanding how the tools perform under differing conditions and scenarios. A thorough assessment of the tools' reliability, scalability, and optimization can be achieved by examining the performance impact across multiple combinations of tree densities and camera angles. Each scene featured two different pine assets, with the estimated total asset count for each density level as follows:

- Low density: ~20,000 assets.
- Medium density: ~50,000 assets.
- High density: ~80,000 assets.

Figure 10 provides a visual representation of each density.

Figure 10. Differences in density (Low, Medium, High) with PCG.

The selection of each camera angle was deliberate for performance testing. Performance impact from trees positioned behind the camera can be assessed by comparing the corner, centre, and rotating angles. With fewer nearby trees, the corner angle is anticipated to yield better results than the other angles. The centre and rotating angles are situated in the same location, offering insights into whether the rotating camera incurs additional performance costs by reprocessing newly visible trees. The moving camera, conversely, tests the effect of rendering culled trees. Comparing the rotating and moving angles allows for an analysis of the differences between reprocessing previously rendered trees and rendering newly visible ones, thereby providing insight into how each tool manages scene complexity and resource utilization.

Test Automation and Procedure

Data collection was performed using Unreal Engine's CSV profiling tool. The first-person blueprint provided by Unreal Engine was used and modified to automate the data collection process and ensure consistency. This blueprint was adjusted to include a 5-second delay before data collection commenced, allowing the scene to load fully and preventing initial performance spikes from distorting the results. Boolean variables, representing true or false values, were incorporated to control whether the camera needed to rotate or move, permitting the camera to commence immediately in the desired motion once the test began. A timer was also implemented to guarantee that each test sample lasted precisely 30 seconds.

Performance Metrics

The performance tests conducted in this study used data gathered from Unreal Engine’s CSV profiling tool. This tool records various performance metrics during runtime and offers detailed insights into different aspects of system performance. Table 1 summarizes the collected metrics along with brief explanations of their significance, serving as a reference for understanding how each metric contributes to the overall performance assessment.

Table 1. Collected Performance Metrics and Their Descriptions.

Stability and Performance:

Metric	Description
Hitches/Min	Number of performance stutters or "hitches" per minute.
HitchTimePercent	Percentage of time spent experiencing performance hitches.
MVP60	Percentage of frames rendered at or above 60 FPS.

Frame Time and Thread Performance:

Metric	Description
Frametime.Avg	Average time taken to render a frame.
GameThreadTime.Avg	Average time spent on the game thread per frame.
RenderThreadTime.Avg	Average time spent on the render thread per frame.
RHIThreadTime.Avg	Average time spent on the RHI thread per frame.
FPS from Frametime	Calculated FPS based on the average frame time.

GPU and Resource Utilization:

Metric	Description
GPUtime.Avg	Average time spent by the GPU per frame.
RHI/Drawcalls.Avg	Average number of draw calls per frame.
MemoryFreeMB.Min	Minimum amount of memory available during testing.
PhysicalUsedMB.Max	Maximum physical memory used during testing.

9.2 Test Results

The test results analysis centres on values that show notable changes, as indicated by the “Notable Change Threshold” chart in Table 2.

The analysis also emphasizes performance consistency, mainly when one tool consistently performs even slightly better than the other across all tests. Results were collected from three separate test batches conducted on different dates, as outlined in the Test Setup section, where one of the test batches involved applying distance culling to all assets to determine if culling produces different results.

Table 2. Notable Change Threshold Reference: Indicates whether a lower (-) or higher (+) value is preferable for each metric.

Noteable Change Threashold	
Hitches/Min	-1
HitchTimePercent	-0,2%
MVP60	-5%
Frametime.Avg	-2 ms
GameThreadtime.Avg	-1 ms
RenderThreadtime.Avg	-1 ms
RHIThreadTime.Avg	-0,5
GPUtime.Avg	-2 ms
RHI/ Drawcalls.Avg	-10
MemoryFreeMB.Min	+50 MB
PhysicalUsedMB.Max	-50 MB
FPS from Frametime	+3 FPS

Table 3 shows performance data using the CSV Profiling tool, with key metrics like frame time, hitching behaviour, and memory usage.

Foliage is shown in blue, and PCG in orange. Longer lines of blue or orange mean that one tool consistently outperformed the other, with darker shades representing more significant differences. Single marks of green indicate occasional better result, where dark green means a notable change and light green means a minor change.

Table 3. Performance data showing key metrics for Foliage (blue) and PCG (orange), with colour coding indicating the extent of performance differences.

	(Foliage - PCG)		(PCG - Foliage)		(Culling)		
All Cams. LOW Avg.	FOL	PCG	FOL	PCG	FOL	PCG	
Hitches/Min	0	0	0	0	0	0	Low
HitchTimePercent	0.0	0.0	0.0	0.0	0.0	0.0	Low
MVP60	47	47	47	47	46	46	Low
Frametime.Avg	32	32	32	32	31	31	Low
GameThreadtime.Avg	3.7	3.6	3.7	3.7	3.7	3.7	Low
RenderThreadtime.Avg	7.3	*8.9*	1.5	*5.9*	5.9	*6.5*	Low
RHIThreadTime.Avg	6.8	6.9	6.8	6.8	6.8	6.8	Low
GPUtime.Avg	30	30	30	30	30	30	Low
RHI/ Drawcalls.Avg	**181**	*183*	**181**	*183*	**181**	*183*	Low
MemoryFreeMB.Min	8559	8666	8686	8611	8540	8560	High
PhysicalUsedMB.Max	3169	3118	3116	3129	3112	3099	Low
FPS from Frametime	32	32	31	32	32	32	High

	(Foliage - PCG)		(PCG - Foliage)		(Culling)		
All Cams. MED Avg.	FOL	PCG	FOL	PCG	FOL	PCG	
Hitches/Min	0	0	0	0	0	1	Low
HitchTimePercent	0.0	0.0	0.1	0.0	0.0	0.0	Low
MVP60	62	62	62	62	62	61	Low
Frametime.Avg	45	*44*	45	*44*	44	*43*	Low
GameThreadtime.Avg	3.7	3.6	3.6	3.7	3.5	3.5	Low
RenderThreadtime.Avg	6.4	8.8	12.6	5.6	16.7	18.6	Low
RHIThreadTime.Avg	6.8	6.8	6.8	6.8	6.9	6.9	Low
GPUtime.Avg	43	42	43	43	43	42	Low
RHI/ Drawcalls.Avg	181	181	180	181	181	181	Low
MemoryFreeMB.Min	8170	8309	8269	8388	8146	8227	High
PhysicalUsedMB.Max	3579	3585	3658	3625	3639	3521	Low
FPS from Frametime	22	23	22	23	23	23	High

	(Foliage - PCG)		(PCG - Foliage)		(Culling)		
All Cams. HIGH Avg.	FOL	PCG	FOL	PCG	FOL	PCG	
Hitches/Min	0	0	1	0	0	0	Low
HitchTimePercent	*28.8*	**20.2**	*29.7*	**15.2**	*19.3*	**13.5**	Low
MVP60	69	69	69	69	69	69	Low
Frametime.Avg	55	55	56	55	54	54	Low
GameThreadtime.Avg	3.8	3.8	3.7	3.5	3.7	3.8	Low
RenderThreadtime.Avg	*9.4*	**2.9**	*8.9*	**3.8**	0.0	4.9	Low
RHIThreadTime.Avg	6.9	6.8	6.9	6.9	6.8	6.8	Low
GPUtime.Avg	53	53	54	53	52	52	Low
RHI/ Drawcalls.Avg	181	182	181	181	181	181	Low
MemoryFreeMB.Min	*7911*	**8029**	*7804*	**7913**	7965	7930	High
PhysicalUsedMB.Max	4009	3921	4093	4047	4043	4044	Low
FPS from Frametime	18	18	18	18	18	19	High

Impact of Asset Density

Both tools maintained similar low-density performance levels, with metrics remaining well within acceptable ranges. As asset density increased, the PCG Plugin demonstrated more stable performance, with average frame time and GPU processing time rising modestly while staying under critical thresholds. This stability was especially evident in metrics like MemoryFreeMB.Min and HitchTimePercent, where the PCG Plugin consistently outperformed the Foliage Tool.

The Foliage Tool showed slightly enhanced results in low- and medium-density environments, particularly in Render Thread times. However, it experienced more substantial performance degradation in high-density scenarios, with metrics such as Hitches/Min and Frametime.Avg more frequently exceeding the desired thresholds, indicating reduced smoothness and responsiveness.

Effect of Camera Angles

Static Angles (Corner and Centre): Both tools performed comparably under static conditions, with only minor differences across most metrics.

Dynamic Angles (Rotating and Moving): Dynamic tests revealed more significant performance differences. The PCG Plugin maintained more stable thread and GPU times, while the Foliage Tool's metrics, especially Hitches/Min and GPUtime.Avg displayed more notable spikes, suggesting that the Foliage Tool may be less optimized for scenes with rapid perspective changes.

Metric-Specific Performance

Hitches and Hitch Time: The PCG Plugin consistently recorded fewer hitches per minute and lower hitch time percentages than the Foliage Tool. This difference was especially noticeable in high-density and dynamic conditions, where the PCG Plugin delivered smoother performance than the Foliage Tool.

Frame time and GPU Time: The PCG Plugin’s frame time and GPU time remained closer to baseline values across most tests. In contrast, the Foliage Tool exhibited increased values that occasionally exceeded the 2ms threshold, particularly in high-density scenes with dynamic camera angles.

Resource Management: While the PCG Plugin excelled in dynamic processing and frame stability, the Foliage Tool occasionally displayed better efficiency in resource-related metrics such as MemoryFreeMB and RHI/Drawcalls. This comparison suggests that while the Foliage Tool can manage low-level resources more effectively in less complex scenes, its performance may diminish as scene complexity increases.

10 Discussion and Comparison

The comparison between the Foliage Tool and the PCG Plugin highlights fundamental differences in workflow, control, and performance optimization. The Foliage Tool offers a hands-on, brush-based approach that suits artists seeking direct placement control, whereas the PCG Plugin provides a rule-based system for procedural asset generation. While PCG excels in automation and large-scale efficiency, the Foliage Tool remains a powerful choice for detailed and intentional placement.

10.1 Usability and Workflow

The Foliage Tool integrates seamlessly with Unreal Engine's interface, offering an intuitive brush-based workflow that lets users paint foliage directly onto surfaces. This method provides immediate visual feedback and is accessible to users of all experience levels. The tool requires minimal setup, enabling users to populate environments quickly without complex configurations. Its straightforward controls for selecting, painting, and erasing foliage make it a practical choice for artists who prioritize direct manipulation and manual precision. The settings are easily accessible in the toolbar and can be adjusted through checkboxes and fine-tuning values, ensuring a streamlined workflow. However, its reliance on manual placement can be time-consuming for larger and diverse environments, as users must individually adjust each asset to achieve the desired variation and density.

In contrast, the PCG Plugin introduces a node-based procedural workflow, requiring a different approach to content generation. Users set rules within PCG graphs, which are applied to specified volumes. Working with graphs requires familiarity with node-based systems and procedural techniques, as users must define and connect rules to control asset placement. While the system offers flexibility and automation, it has a steeper learning curve, especially for those unfamiliar with procedural logic. Setting up the system can be complex, as

users must structure and connect rules to ensure effective asset placement. If the procedural rules are not well-optimized, they may lead to performance issues, such as excessive asset spawning or inefficient calculations. However, once configured, the system automates asset generation, reducing the need for manual adjustments. This automation streamlines workflows, enabling users to handle larger projects and greater scalability easily. Though the initial setup can be demanding, the PCG Plugin ultimately offers a more efficient workflow for complex environments once mastered.

10.2 Customisation and Control

When comparing customization and control, the Foliage Tool provides a straightforward workflow with fixed settings that simplify asset placement. These predefined options offer accessible controls for adjusting elements such as brush size, density, rotation, scale, terrain alignment, collision settings, and culling distance. This simplicity makes the tool user-friendly and well-suited for manual adjustments, giving artists precise control over individual assets.

Although the tool's customization capabilities are limited to these predefined settings, it does offer some post-placement flexibility. The Select Tool allows users to manually adjust individual assets after placement, while the Reapply Tool can modify selected settings across multiple assets without requiring individual adjustments. Despite these features, the tool remains limited to manual corrections and does not support global rule changes or dynamic updates. Additionally, the lack of automated placement methods can make scalability challenging in larger environments.

Nevertheless, the Foliage Tool provides precise artistic control over individual assets, making it particularly useful for crafting smaller, highly detailed environments where fine-tuned placement is essential.

In contrast, the PCG Plugin provides greater flexibility through its node-based system, allowing users to define detailed placement rules, apply constraints, and filter techniques to dynamically control asset distribution. Users can create

complex procedural systems by adding as many nodes as necessary, offering complete control over randomization, placement rules, and asset distribution.

This procedural approach enables efficient large-scale changes, with adjustments to rules immediately reflected across the affected environment. The PCG Plugin also supports runtime adjustments, allowing environments to adapt dynamically based on game conditions.

To further expand customization, the PCG Plugin supports custom nodes, which users can create using Blueprints or C++. These custom nodes allow developers to build specialized logic for asset distribution, offering precise control over procedural placement.

Moreover, the PCG Plugin includes powerful constraints such as minimum and maximum density limits, slope restrictions, and proximity rules. These constraints help ensure that procedural placement remains controlled and realistic, reducing the risk of unnatural patterns.

Despite these advantages, the procedural approach may limit fine artistic control compared to the manual precision offered by the Foliage Tool. Without careful rule management, PCG-generated environments can sometimes develop repetitive or artificial-looking patterns, requiring additional refinement to achieve a more convincing result.

This contrast underscores the fundamental difference between the two tools: the Foliage Tool prioritizes manual precision and predictable results, while the PCG Plugin emphasizes flexibility and automation, making it better suited for large-scale or dynamic environments.

10.3 Documentation and Support

The Foliage Tool is supported by extensive documentation and a strong community. Its long-standing presence in Unreal Engine has resulted in numerous tutorials, guides, and discussions that assist users in navigating its functionalities. However, official video documentation has not been updated

recently, which means older tutorials may not fully cover newer features. The PCG Plugin, being a more recent addition, has continuously evolving documentation. Epic Games provides frequent updates, including video tutorials and step-by-step guides that help users understand its procedural workflow. While the community is still growing, the increasing resources and discussions suggest that the plugin will become more accessible over time.

10.4 Realism and Output

When evaluating realism, the Foliage Tool and PCG Plugin demonstrated the ability to generate convincing Finnish mixed forests. However, the differences in their placement methods led to distinct visual results. The Foliage Tool's randomized placement created an organic, uneven distribution that closely resembled the natural variability in real ecosystems. This approach worked particularly well for depicting undergrowth and scattered tree arrangements, allowing for irregular gaps and clusters. However, this randomness also introduced inconsistencies in density, making it difficult to maintain precise control over spacing. Despite manual adjustments, some areas appeared sparse while others became unintentionally overcrowded, necessitating iterative tweaks to achieve a balanced look. In contrast, the PCG Plugin's structured, rule- based placement system provided more predictable results. By defining specific spacing parameters and constraints, the PCG- generated forests maintained a consistent distribution, which was especially beneficial for tree placement. This method proved effective for achieving the characteristic structure of a Finnish mixed forest, where Scots Pine dominates with spaced-out formations while Spruce and Birch add variety in a controlled manner. However, the procedural nature of PCG also introduced a certain uniformity that, if not carefully adjusted, risked creating patterns that appeared artificial. To counteract the unintended clustering, additional randomness parameters were introduced into the settings to enhance the organic feel of the forest. One key distinction in realism was the handling of asset density. The Foliage Tool's manual painting process allowed for precise placement, but adjusting asset

distribution after initial placement posed challenges. Increasing the brush size to cover larger areas unintentionally altered the density, requiring multiple adjustments to maintain a natural look. Conversely, the PCG Plugin facilitated automatic redistribution when parameters were adjusted, minimizing the need for manual repositioning. This automatic redistribution helped maintain a realistic balance between tree species and avoided unintended clustering. Ultimately, while both tools could produce realistic results, their suitability depended on the intended level of control and the specific aesthetic goals of the project. The Foliage Tool excelled in artistic flexibility and intuitive placement but required more manual intervention to refine the final look. The PCG Plugin offered greater consistency and efficiency, particularly for large- scale environments, though achieving a natural appearance required fine- tuning of procedural randomness. For the specific goal of recreating a Finnish mixed forest, the PCG Plugin's structured approach proved more effective in maintaining realistic species distribution and density while minimizing manual adjustments.

10.5 Performance

The performance tests revealed that while the PCG Plugin and the Foliage Tool performed similarly overall, their strengths varied depending on scene complexity. The PCG Plugin demonstrated better performance in memory management and in reducing hitches during high-density scenes. Metrics such as MemoryFreeMB.Min and HitchTimePercent reflect this advantage, with the PCG consistently outperforming the Foliage Tool.

In contrast, the Foliage Tool showed slightly better results in low- and medium-density environments, particularly in Render Thread times.

Camera angles also influenced performance outcomes. As expected, the corner camera generally produced better results due to fewer visible assets. The rotating camera exerted more strain on performance than static angles, likely due to the constant reprocessing of newly visible trees. Despite this added

strain, the PCG's improved performance in high-density scenes remained evident.

These results suggest that the PCG Plugin may be preferable for larger, more complex scenes, while the Foliage Tool might provide a slight efficiency advantage in less demanding environments. The optimal choice depends on the project's requirements and expected performance conditions.

11 Conclusion

This thesis explored the strengths and weaknesses of Unreal Engine's PCG Plugin and the Foliage Tool in the context of generating realistic game environments, specifically focusing on recreating a Finnish mixed forest. Through detailed hands-on testing and performance evaluations, the study demonstrated that while both tools can produce visually convincing results, they differ significantly in workflow, control, and scalability.

The results showed that the PCG Plugin excels in automated, large-scale generation and performance stability under high-density conditions. Its structured, node-based workflow supports runtime updates and enables consistent asset placement, making it especially effective for expansive or dynamic environments. In contrast, the Foliage Tool offers a more intuitive and artist-friendly interface, allowing for precise manual placement that supports detailed artistic direction. However, it becomes less efficient in large or complex scenes.

These findings are significant as they provide valuable insights for game developers and technical artists. By outlining each tool's practical benefits and constraints, this research supports informed decision-making based on specific project needs, whether those favour automation and scalability or manual control and precision.

However, the study comes with limitations. The scope was limited to forest generation within Unreal Engine, with a focus on specific biome- and asset packs. The performance tests, while extensive, were confined to static conditions and may not fully reflect dynamic runtime environments in actual gameplay. Additionally, the PCG Plugin is still under development, which means future updates could impact its capabilities.

Despite these constraints, the research highlights valuable trade-offs between procedural and manual workflows. The results can be applied in game development scenarios that demand either efficiency or fine-grained control

over environmental design. Beyond game development, the insights may also be relevant to simulations, virtual production, and architectural visualization where environmental realism and system performance are key factors.

Future work could expand the study by incorporating more biomes, testing runtime procedural updates during gameplay, or exploring hybrid workflows that combine both tools. As the PCG Plugin continues to evolve, revisiting its performance and features in future versions would offer a richer understanding of its long-term viability and potential as a core tool in real-time content creation.

References

Aversa, D. (2015, May). *Procedural Contents Generation: History and Techniques used in modern video-game industry.* Retrieved 09 30, 2024, from https://www.davideaversa.it/wp-content/uploads/2015/06/Procedural-Contents-Generation.pdf

Black, C. (n.d.). *A Deep Dive into Realism in Landscape Painting: What It Is and How to Master It*. Retrieved 10 25, 2024, from https://www.chuckblackart.com/blogs/the-painters-block/realism-in-landscape-painting-guide

Deacon, D. P. (2024). *Procedural Content Generation (PCG) in a Nutshell*. Retrieved 01 14, 2025, from https://medium.com/@deaconline/procedural-content-generation-pcg-b54f4c1959cd

DiLaura, J. (2023). *A Look at Unreal Engine Procedural Generation of Content*. Retrieved 04 15, 2024, from https://interactiveimmersive.io/blog/unreal-engine/a-look-at-procedural-content-generation-in-unreal-engine/

Epic Games. (2011a). *Epic Games Releases June 2011 Unreal Development Kit Beta*. Retrieved 08 30, 2024, from https://www.unrealengine.com/en-US/blog/epic-games-releases-june-2011-unreal-development-kit-beta

Epic Games. (2011b). *UDK | Foliage*. Retrieved 08 28, 2024, from https://docs.unrealengine.com/udk/Three/Foliage.html

Epic Games. (2021). *Foliage Tool*. Retrieved 03 01, 2024, from https://dev.epicgames.com/documentation/en-us/unreal-engine/foliage-tool?application_version=4.27

Epic Games. (2023). *Unreal Engine 5.2 is now available!* Retrieved 04 15, 2024, from https://www.unrealengine.com/en-US/blog/unreal-engine-5-2-is-now-available

Epic Games. (2024a). *Blueprints Visual Scripting*. Retrieved 09 25, 2024, from https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine?application_version=5.4

Epic Games. (2024b). *Features*. Retrieved 09 07, 2024, from https://www.unrealengine.com/en-US/features

Epic Games. (2024c). *Foliage Mode*. Retrieved 08 28, 2024, from https://dev.epicgames.com/documentation/en-us/unreal-engine/foliage-mode-in-unreal-engine

Epic Games. (2024d). *Procedural Content Generation Framework*. Retrieved 04 15, 2024, from https://dev.epicgames.com/documentation/en-us/unreal-engine/procedural-content-generation--framework-in-unreal-engine

Epic Games. (2024e). *Procedural Content Generation Overview*. Retrieved 08 16, 2024, from https://dev.epicgames.com/documentation/en-us/unreal-engine/procedural-content-generation-overview?application_version=5.5

Epic Games. (2024f). *Programming and Scripting*. Retrieved 09 16, 2024, from https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-programming-and-scripting

Epic Games. (2025a). *Unreal Engine*. Retrieved from Unreal Engine: https://www.unrealengine.com/en-US

Epic Games. (2025b). *Unreal® Engine End User License Agreement*. Retrieved 04 15, 2024, from https://www.unrealengine.com/en-US/eula/unreal

Epic Games. (n.d.). *Occulusion Culling*. Retrieved from Unity Documentation: https://docs.unity3d.com/Manual/OcclusionCulling.html

Erolin, J. (n.d.). *What Is Unreal Engine?* Retrieved 04 15, 2024, from https://www.bairesdev.com/blog/what-is-unreal-engine/

Game-Ace. (2024a). *The Best Ways to Use Procedural Generation in Games*. Retrieved 09 13, 2024, from https://game-ace.com/blog/procedural-generation-in-games/

Game-Ace. (2024b). *What is The Key to Immersive Game Design?* Retrieved 10 25, 2024, from https://game-ace.com/blog/immersive-game-design/

Hai, T. H. (2022). *Game Development with Unreal Engine.* Retrieved 04 15, 2024, from https://www.theseus.fi/bitstream/handle/10024/751778/Ha%20Hai%20.pdf?sequence=2

Karppinen, A. (2023, October). *Creating Foliage for Video Games.* Retrieved from theseus: https://www.theseus.fi/bitstream/handle/10024/808866/Karppinen_Aleksi.pdf

Latif, A., Zuhairi, M. F., & Qudus Khan, F. (2022). A Critical Evaluation of Procedural Content Generation Approaches for Digital Twins. *Journal of Sensors, 2022*(https://doi.org/10.1155/2022/5629645), 13. Retrieved from https://onlinelibrary.wiley.com/doi/10.1155/2022/5629645

Li, D. (n.d.). *C++ vs Java: Unraveling the Key Differences for Developers*. Retrieved 09 16, 2024, from https://aloa.co/blog/c-plus-plus-vs-java#

Logut, A. (2024). *Introduction to Procedural Generation plugin in UE5.4*. Retrieved 08 16, 2024, from https://dev.epicgames.com/community/learning/tutorials/j4xJ/unreal-engine-introduction-to-procedural-generation-plugin-in-ue5-2

Madigan, J. (2010). *The Psychology of Immersion in Video Games*. Retrieved 10 25, 2024, from https://www.psychologyofgames.com/2010/07/the-psychology-of-immersion-in-video-games/

Meegle. (2024). *Procedural Content Generation*. Retrieved 02 05, 2025, from https://www.meegle.com/en_us/topics/gaming/procedural-content-generation

Project Nature. (2024, September 18). *Temperate Vegetation: Spruce Forest*. Retrieved from Unreal Engine | FAB: https://www.fab.com/listings/f8044501-17a2-498f-b198-5f1bc71ee87a

Realpixels Studio. (2024, October 1). *Nordic AutoBiome.* Retrieved from Unreal Engine | FAB: https://www.fab.com/listings/e9be5572-841f-4bb4-9e42-8aa36384062b

Risi, S., & Togelius, J. (2019, November 29). *Procedural Content Generation: From Automatically Generating Game.* Retrieved 09 13, 2024, from https://arxiv.org/pdf/1911.13071v1

Smith, G. (n.d.). *An Analog History of Procedural Content Generation.* Retrieved 09 28, 2024, from http://www.fdg2015.org/papers/fdg2015_paper_19.pdf

Sweeney, T. (2014). *Welcome to Unreal Engine 4*. Retrieved 09 09, 2024, from https://www.unrealengine.com/en-US/blog/welcome-to-unreal-engine-4

Sweeney, T. (2015). *If You Love Something, Set It Free*. Retrieved 09 09, 2024, from https://www.unrealengine.com/en-US/blog/ue4-is-free

Toolfarm. (2024). *A Guide to Realistic Digital Vegetation Creation for 3D Landscapes*. Retrieved 10 25, 2024, from https://www.toolfarm.com/tutorial/vegetation/

Unreal Engine. (2011). *Unreal Engine YouTube Channel*. Retrieved 02 15, 2025, from https://www.youtube.com/@UnrealEngine

Unreal Engine. (2017, February 14). *Getting Started with Landscapes Materials and Foliage | Live Training | Unreal Engine*. Retrieved from YouTube: https://www.youtube.com/watch?v=PDlGKZ1c3Zc&ab_channel=UnrealEngine

Valtsa, L. (2020). *Overview of Mixed Forests in Finland: EuMIXFOR Country Report.* Retrieved from University of Helsinki: https://tuhat.helsinki.fi/ws/portalfiles/portal/85605360/EuMIXFOR_CountryReport_FI.pdf

Van Brummelen, J., & Chen, B. (n.d.). *Procedural Generation: Creating 3D worlds with Deep Learning*. Retrieved 09 13, 2024, from https://web.archive.org/web/20200417084352/https://www.mit.edu/~jessicav/6.S198/Blog_Post/ProceduralGeneration.html

Wikipedia. (2024, September 5). *Unreal Engine*. Retrieved 04 15, 2024, from Wikipedia: https://en.wikipedia.org/wiki/Unreal_Engine

Wikipedia. (2025, January 26). *Tim Sweeney*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Tim_Sweeney

Wirtz, B. (2023). *The Power of Experience: The Wonders of Video Game Immersion*. Retrieved 10 25, 2024, from https://www.gamedesigning.org/learn/game-immersion/