Procedural Generation of Vegetation Around Geometry

by Mikolaj Debiec

Research Report

Submitted in partial fulfilment for the degree of
BSc (Hons) Computer Science

Supervised by Dr. Babis Koniaris

Heriot-Watt University
School of Mathematical and Computer Sciences

20th November 2025

Authorship Declaration

Course code and name	F20PA/B/C - Honours Project
Type of assessment	Individual
Coursework title	Procedural Generation of Vegetation Around Geometry
Student name	Mikolaj Debiec
Student ID	H00399881

Student signature: Mikolaj Debiec

Date: 20th November 2025

Abstract

Adding on-surface decoration such as vines to a model is often important in game design and 3D art in order to add realism or invoke a certain atmosphere. It can, however, be a tedious process if manually modelling the decoration. In this proposal, a way to procedurally generate models resembling plants, particularly vines, on top of an already existing mesh in preset, modifiable patterns is presented. The approach will use shape grammars or graph grammars to produce the vegetation based on the mesh that it inhibits to create a model that is continuous, clean and appropriate for the model that it sits on.

Table of Contents

1 Introduction

1.1 Background and Motivation

When creating convincing 3D scenes, decoration is a crucial step in the process. It both creates visual interest and can make the scene more realistic. An important aspect of that is producing models on the surface of other such as moss or ivy, but when done manually, it can be a long and tedious process which can involve a lot of small adjustments to make sure the models are congruent with each other.

When it comes to models such as vines that grow in patterns, it becomes even more time-consuming to do them manually. While this can be a non-issue in small projects, it remains a major problem in games, where many models may have to be made with a similar theme. A better method to generate these models on existing meshes is procedurally, which is proposed in this document, making the process fast and consistent, as well as modular for dynamic use within game development and 3D art projects.

1.2 Aim and Objectives

The main objective of the project is to implement a tool that takes an existing mesh and procedurally generates a vine on the surface of it based on a point or a line chosen by the user. The generation should also be modular, with different aspects of it able to be changed before generation. Unlike the manual process, the tool should be fast, consistent, and its results easy to modify.

Key requirements for the tool should include:

- Procedural generation along the surface which works in the context of each specific mesh that is, it is congruent with the mesh
- Modifiable generation which should include different patterns of the vine, density of branches, length from the starting point or line
- The generation should take at most several minutes
- The vine should be and look continuous (as in, it should not be disjointed and should be either part of the original model or a new model on the surface and look accordingly)

Other objectives include:

- Developing the tool iteratively while reevaluating requirements that were met during each development cycle
- Designing an appropriate way to test the features of the tool based on different meshes and changing different aspects
- Designing an evaluation to compare the results to existing work on the subject

1.3 Organisation

This document is split into 5 main sections:

(1) Introduction
This section - consists of the background and motivation of this project, the objectives that this project aims to achieve and this subsection which details what each major section consists of
(2) Background Research
The background research section details the research undertaken as part of the preparation for the project and conclusions drawn from it. This involves research into procedural generation techniques, the software and tools to be used in this project and previous work on the topic.
(3) Work Plan
This section aims to be a more direct way to see exactly what needs to be done for the project to succeed. This includes the requirements of the system and the tasks needed to be complete for the project to be considered finished. It also explains the evaluation strategy.
(4) Feasibility
This section shows what steps have already been undertaken to prepare for the projects which includes the setup of the workspace and the prototype made to show that the work is feasible and to prepare for the work.
(5) Conclusion
Summarises the key points of the rest of the document. But especially the motivations and goals as well as the proposed work in a more concise manner.

2 Background Research

This section will explore different types of procedural generation (2.1), tools that will be used in the project (2.2) and previous work on the topic (2.3) concluding with the summary (2.4)

2.1 Procedural Generation

As the work will be mainly rooted in procedural generation, it is important that it is established what procedural generation is first. This will be followed by a description of procedural generation techniques and ways in which they are used. The final choice between them will be decided in the implementation of the project.

Procedural Generation
Procedural generation is the process of generating something (in this case, a 3D model) using an algorithm. This is often used in game development [24], for example the world in *Minecraft* is created procedurally[29]. A procedural generation algorithm can be made using many methods[14], the one chosen depends heavily on what is being generated [4]. In the case of this project grammars will be used to create the algorithm.

Formal Grammars
Formal grammars are made up of a set of symbols and a set of rules to form them into

some set of strings[2]. This set of strings is called a language. In other words, a grammar can be thought of as a way to generate languages. Because of their capacity to explore syntax and how it affects a language, formal grammars are extensively used in linguistics, computing and many other fields [11] to analyse and evaluate things from programming languages [28] to music[25]. Their foundation can also be formed into new grammars for more specific purposes as will be demonstrated in the following sections.

Shape Grammars, Graph Grammars and L-Systems
In the same way that a formal grammar can modify a set of symbols to create strings, other grammars can be used to create various things. These grammars can come in many forms, which can be a model of modifying and exploring different topics.

Shape grammars take as an input shapes and their rules determine which way the said shapes can be transformed[20]. Although originally designed for the classical arts such as sculpture[26], they have naturally become invaluable in the digital world. Currently, shape grammars are widely used in game development [12], architecture [10] and civil engineering [21], most commonly in the form of procedural generation where they can be used to make a variety of assets from only basic building blocks.[19][9].

Similarly, graph grammars take in graphs and use rules to transform them into different ones[7]. While initially created for image processing[8], graph grammars commonly focus on mathematics [16], Artificial Intelligence [15] and even biology [6] as this technique can model a surprising amount of phenomena and explore how they change. Most recently, graph grammars have been increasingly used for procedural generation, such as in Merrell [18] and Vatresia et al. [27] as its simple representation can result in deceptively complex and adaptable generation.

Finally, L-Systems are a formal grammar [13] initially invented in order to emulate multicellular organisms [23], they can be used in the context of procedural generation specifically when generating plants [22] to create believable, natural generations.

2.2 Software and Tools

The main tool that will be used in this project is the Unity game engine. Because graphics have to be integrated with code and user input, a game engine is a simple way to ensure the required capabilities are present without building graphics from scratch. Unity was selected as a solution that doesn't have too many unnecessary features. The other popular option is Unreal Engine which has a much higher scope unnecessary for this project and more fit for a full game. [5] Additionally, Unity is a faster of the two engines [1] - and for a proof of concept where rendering, lighting etc. is not important Unity is a much better solution.

The programming language used for the project is C#. The game engine I'm using supports it very well, it's also a well-suited language for computer graphics in general. The IDE used is VSCode for its simplicity, as well as for its addons that help in debugging.

The project pertains to the algorithm used to procedurally generate the model based on the mesh under it. However, the actual cosmetic look of the model won't be procedural - the 3D software Blender will be used to make the model for this purpose. However, if for any reason it is difficult to work with models that were made, pre-made assets such as from the Unity Asset Store may be used.

2.3 Previous Work

Previous work on the topic of procedural generation using shape or graph grammars is quite extensive. For example, Merrick et al. [19] shows that shape grammars as a technique can be used for creating many assets in MMO games and Freiknecht [9] uses it for modelling cities in a driving game. Similarly, Merrell [18] talks about using graph grammars for procedural modelling specifically.

These papers exemplify that procedural generation using these techniques is common and feasible but research was needed into work more similar to the planned product. The first paper found, Li et al. [17] focused on producing patterns on meshes using shape grammars. This was particularly useful as one requirement of the project is to be able to produce patterns of vines which was not only shown to work using a shape grammars, but work so on detailed and complex shapes. Finally, Buron et al. [3] used a very similar method to what is planned for the project - modelled vines on top of meshes, worked on similarly with a shape grammar, however as opposed to the other project this is also dynamic and done in real time, unlike this proposed project.

The project's results will most likely be compared for evaluation both to (Buron et al. [3]) and (Li et al. [17]). Although they differ in some ways (real time vs pre-made pattern), it can be used to evaluate the way the pattern results on the final model and compare them using different (or similar) meshes.

2.4 Summary

In summary, the project will use either shape or graph grammars to procedurally generate a model on a surface of another mesh. This will be done in the Unity Game Engine using C# and VSCode as the IDE. Either Blender or a pre-made asset may be used for the cosmetic look of the vines.

Previously mentioned research has shown that making procedural models using grammars is widely used. Similar projects have also been undertaken and written about, however, this project aims to be a quick solution for simplicity and doesn't aim to work like a real-time brush but rather a quick generation across the entire model. It also aims to be more modular than similar work that focuses on generating patterns on meshes.

3 Work Plan

This section will detail the proposed work plan for the project. Section 3.1 will detail the functional and non functional requirements described in the MoSCoW system, section 3.2 will outline the tasks involved in the project including its evaluation strategy.

3.1 Requirements

This section details the requirements of project in the MoSCoW system, which classifies their priority based on whether the requirement **Must**, **Should**, **Could** or **Will Not** be included to properly satisfy the project's aims.

System Requirements			
Requirement ID	Requirement	MoSCoW classification	Notes
FR-S01	The System shall be able to generate vines on top of any valid mesh	Must	'Valid' in this context means for the model to be geometrically and topologically sound.
FR-S02	The System's generation of the vines shall be procedural	Must	
FR-S03	The generation shall have modifiable properties	Should	
FR-S03.1	The generation shall have a selection of patterns in which the vines can be generated	Should	

FR-S03.2	The generation shall have a custom starting point chosen by the user	Should	The starting point can be a point or a line
FR-S03.3	The generation shall have adjustable density of branches	Should	
FR-S03.4	The generation shall have an adjustable vine length from the starting point	Should	
NFR-S01	The generation shall take a at most several minutes	Must	
NFR-S02	The generation shall look continuous	Must	’continuous’ meaning non-segmented, like part of one long vine
NFR-S03	The system’s generation should be on top of the ’host’ mesh, respecting its geometry	Must	The generation should not be at any point too high, inside or otherwise incorrectly placed relative to the mesh
NFR-S04	The system’s generation shall not clip into itself	Must	

User Requirements			
Requirement ID	Requirement	MoSCoW classification	Notes
FR-U01	The user shall be able to generate on any valid mesh	Must	'Valid' in this context means for the model to be geometrically and topologically sound.
FR-U02	The user shall be able to select a point on the mesh from which to start generation	Should	
FR-U03	The user shall be able to adjust the properties of generation	Should	
NFR-U01	The properties of generation should be easy to change	Should	
NFR-U02	The interface should have visual feedback when changes to the properties are made	Should	

3.2 Tasks

This section will break down the project into smaller tasks to outline how the work will be completed in a more structured manner.

(1) **Literature Review**
The Literature Review will focus on developing foundational knowledge required to build the project from a more practical aspect. That includes a detailed examination of grammars and how they affect meshes, altering geometry for the purpose of making modifiable properties and a deeper look into the Unity API.

(2) **Prototype Design**
The Prototype Design stage will involve a more detailed approach to outlining the needs of a basic version of the system. This involves an on-paper design of the grammar to be used, basic UI design, a look into what properties will be adjustable and how this can be implemented among other qualities. The prototype design stage

should take into account all key functional requirements.

(3) **Prototype Implementation**
The Prototype Implementation stage will involve the implementation of basic functionalities of the system as designed in task 2. Any difficulties with the project at this stage should be noted and any features too troublesome and time-consuming not implemented unless they are essential for the functionality of the prototype. These difficulties shall then be re-approached and, if necessary redesigned in the next stage.

(4) **Prototype Evaluation**
The Prototype Evaluation stage is a simple evaluation of the prototype implemented in the previous stage. This involves assessing the successes and failures of the prototype as well as ideas for refining them. If any features could not be finished in the prototype, this stage should deconstruct the reasons for that and lead to a redesign or change in approach for that feature. The result of this stage should be a well-defined groundwork for the final design that pre-emptively avoids large issues in the final design.

(5) **Final Design**
The Final Design of the project will be an in-depth plan for the system. Every aspect should be taken into account at this stage, including the functional and non-functional requirements. The evaluation of the prototype should be taken into account at this stage and features revised as needed.

(6) **System Implementation**
Implementation of the final design - this should be the longest stage, it involves building on the foundation laid by the prototype to implement all of the requirements of the system as put in Task 5.

(7) **System Evaluation**
This stage involves evaluating the final system by showcasing its results and comparing them to previous work on the topic. The evaluation strategy is laid out in following steps:

1. The generation is utilized on basic shapes like a sphere, cube and a cylinder and evaluated on its efficacy and adherence to the requirements.

2. The basic shape generation is compared to similar shapes (whenever possible) from previous work on the topic to evaluate its basic usage compared to those.

3. The generation on basic shapes is tested by adjusting various properties, this is to showcase the variety that the method can provide. It is compared to the capabilities of previous work focusing on the flexibility of this project vs the predecessors'.

4. The generation is utilized on complex shapes, similar to the ones used in the results of previous work to further test its requirements.

5. The properties of the generation on complex shapes is adjusted to try to achieve similarity to previous work's results. This is in order to recognise differences when trying to achieve a similar result and how this work differs.

6. The generation is attempted on complex shapes outside of the scope of previous work, in order to ascertain limitations of the project and test its efficacy in practical scenarios.

This section highlighted the work plan and how the full project is going to be undertaken, the next section will focus on the work already completed in preparation for the project.

4 Feasibility

This section will detail the work already done on the project. Section 4.1 will detail the set up of the environment such as the IDE and the engine, as well as touch on assets that will be used in the project. Section 4.2 will talk about the prototype made in preparation of the project.

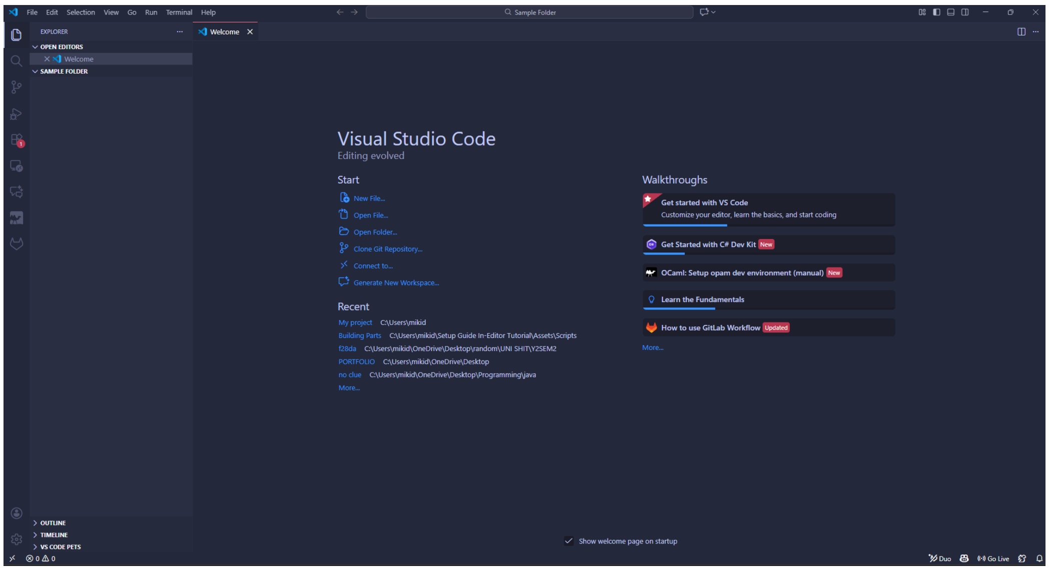
4.1 Coding Environment

4.1.1 IDE & Engine. The coding environment will consist of VSCode as the IDE with addons that are useful for this work. This will be used as an external code editor with Unity as the environment where the results of the code can be seen.

I will use extensions that make the work more efficient and better integrated, they consist of:

- The C# and C# Dev Kit, which allow for easier debugging, testing as well as general integration of the C# language in VSCode. These will smooth out the coding to allow for quick and efficient workflow.

- Unity, which allows for integration of the Unity API into VSCode, this means that things which usually would not flag up as they might be Unity-only errors will now show up in VSCode. It also allows for edited code making quick changes in Unity among many other small features which allow for a cohesive experience.



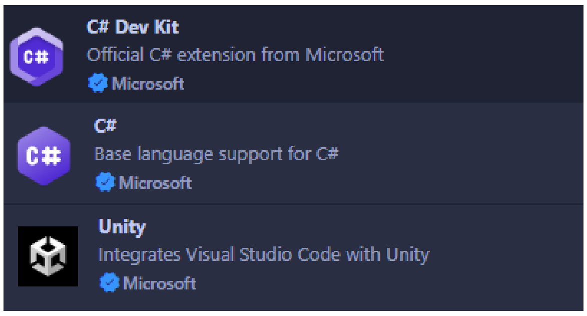
Visual Studio Code
Editing evolved

Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Connect to...
Generate New Workspace...

Recent
My project C:\Users\mikid
Building Parts C:\Users\mikid\Setup Guide In-Editor Tutorial\Assets\Scripts
f28da C:\Users\mikid\OneDrive\Desktop\random\UNI SHIT\Y2SEM2
PORTFOLIO C:\Users\mikid\OneDrive\Desktop
no clue C:\Users\mikid\OneDrive\Desktop\Programming\java
More...

Walkthroughs
Get started with VS Code
Customize your editor, learn the basics, and start coding
Get Started with C# Dev Kit New
OCaml: Setup opam dev environment (manual) New
Learn the Fundamentals
How to use GitLab Workflow Updated
More...

Fig. 1. VSCode setup



C# Dev Kit
Official C# extension from Microsoft
Microsoft

C#
Base language support for C#
Microsoft

Unity
Integrates Visual Studio Code with Unity
Microsoft

Fig. 2. Extensions to help with the workflow

4.1.2 Models and assets. While a model will be required for the project, The process of making it requires a better knowledge of what is needed for the code specifically, otherwise, it could be incompatible. For that reason, a model has not been produced - it depends heavily on the requirements of the program which will be found as the project is being worked on.

Using a pre-produced model is also likely impractical - most vine models are not made for procedural generation, and rather often focus more on the stylistic choice and are more so usable in many scenarios. For this specific project where it has to be taken into account where the surface is, and therefore it is much more logical to create a custom model, once it is known what form it needs to take to satisfy the project.

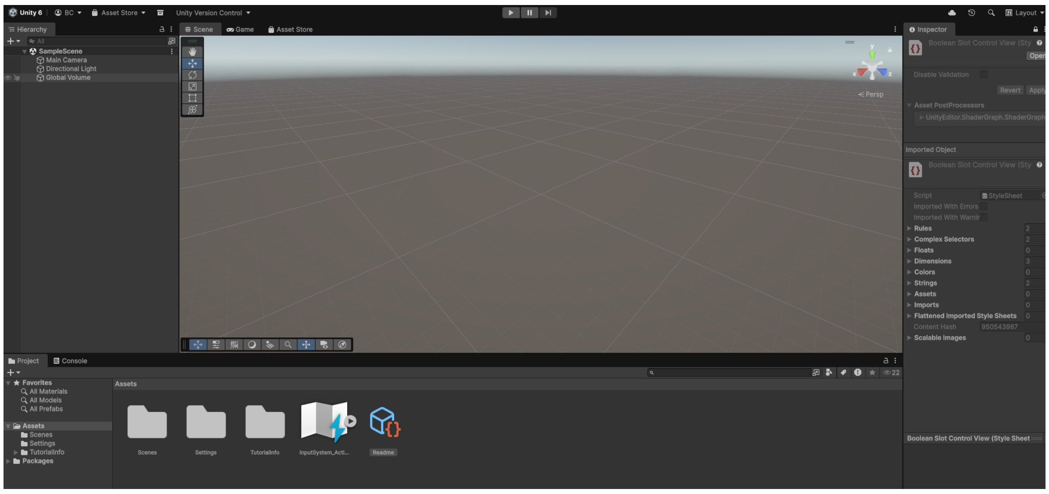


Fig. 3. Unity setup - it uses the latest version, no additional changes should be needed.

4.2 Prototype

As my prototype, to both show that the tools I have picked are appropriate and that my concept is feasible, I have produced a modular procedural generator of buildings in Unity. While it lacks the aspect of patterns on surfaces present in the proposed project it uses a grammar to procedurally generate a building as specified by the user. This involves editable size of the building and changing roofs that are partly random, depending on the shape.

The prototype served the purpose of getting me familiar with Unity's API & It's workflow, as well as using grammars in a practical context. It also strengthened my knowledge of C#, especially in a graphics context. But many properties of the project were also practised via the prototype, as I've presented in the screenshots.

This section both showcased how the workspace setup and presented the modular procedural generation prototype - which exemplified a lot of features of the final project.

5 Conclusion

This section will summarise the key takeaways from this proposal. Section 5.1 concludes main motivations and goals of the project and section 5.2 summarises the work to undertake.

5.1 Motivations and Goals

When making cosmetically good, convincing assets, on-mesh generation is an important aspect, however, when made by hand it can be a long, timely endeavour. Procedural generation also often focuses on making new assets - not editing existing ones, making this a difficult task to automate. This project focuses on making procedural generation on top of surfaces quick, easy and modifiable. This is not only achievable, but also useful in the service of developing 3D assets that are distinct and convincing. As the rest of the document has shown, this is not only useful but also achievable using current and existing procedural

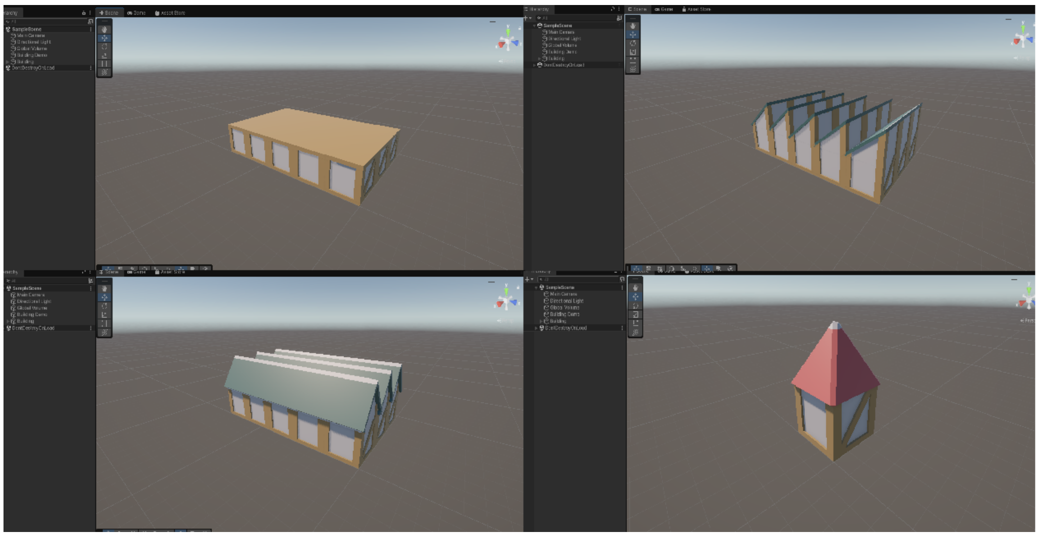
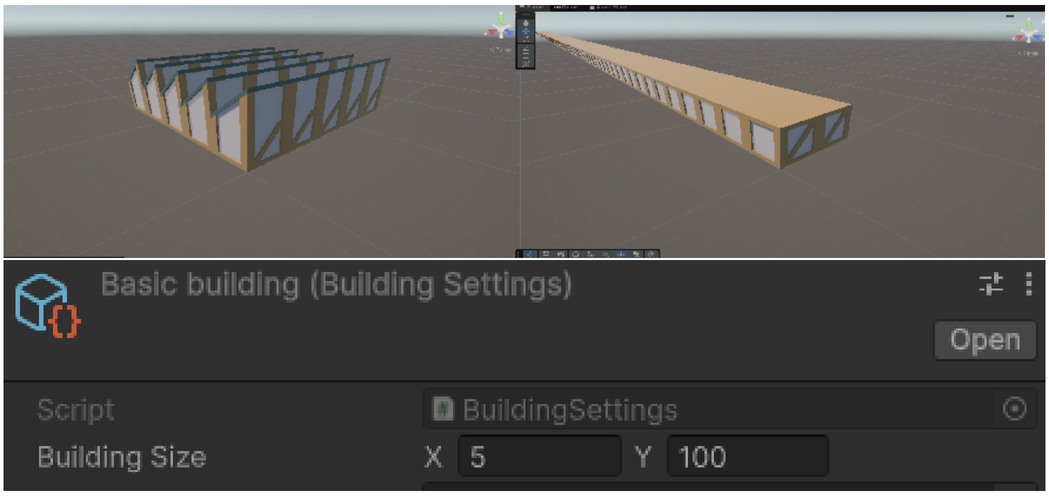


Fig. 4. The Prototype's houses' roofs are random, however if a house is small enough the roof will always be pointed - this was to practice adaptable procedural generation - not just purely random as this will be essential to this project.


Basic building (Building Settings)

Open

Script BuildingSettings

Building Size X 5 Y 100

Fig. 5. To practice the 'modifiable property' of my project, the user can change the size of buildings within the program.

generation techniques.

Unlike other projects on this topic, this project will focus on making a rather quick and easy solution that still functions properly and is a great asset when working on projects with many models.

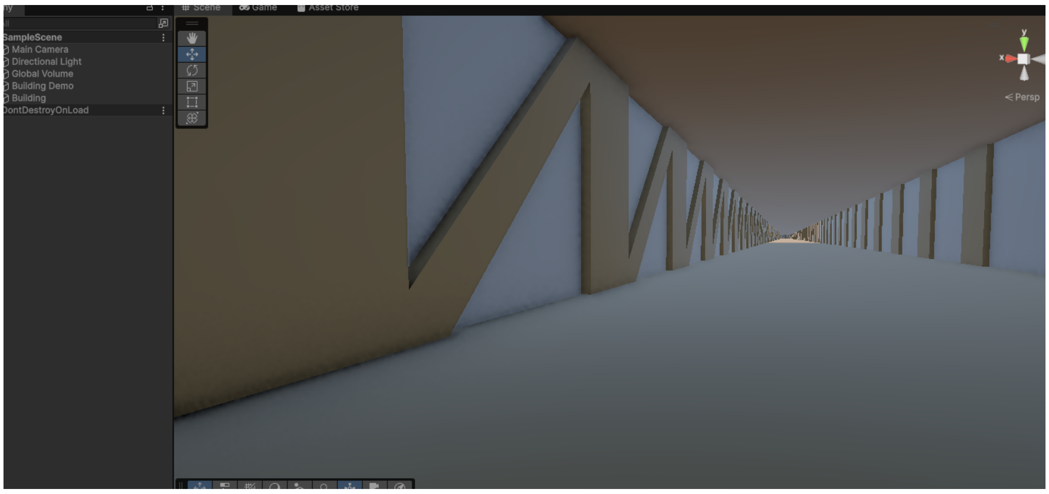


Fig. 6. The prototype isn't just a hack using sets of boxes - the inside is empty, showing that this is fully procedurally generated.

5.2 Proposed Work

This project intends to lead to a creation of a procedural generation tool that uses grammars to procedurally generate a model on the surface of an existing mesh. This should be a quick, easy process for the user that takes into account the geometry to create an appropriate model of vegetation - such as vines that can wrap about any viable mesh. Furthermore, the generation aims to be adjustable to further its efficacy as a tool that can be quickly adjusted and create distinct, interesting assets for even a large amount of assets without the need for adapting the original mesh by the user.

The adapted properties such as density of vines, starting point of the generation, size and length should lead to varied and visually interesting results that can be used on a large amount of models and look distinct on each one. The final result should lead to a useful tool both in video game development and even 3D art.

References

[1] Kamil Abramowicz and Przemysław Borczuk. 2024. Comparative analysis of the performance of Unity and Unreal Engine game engines in 3D games. *Journal of Computer Sciences Institute* 30 (2024), 53–60.

[2] Salbiah Ashaari, Sherzod Turaev, and Abdurahim Okhunov. 2016. Structurally and Arithmetically Controlled Grammars. *International Journal on Perceptive and Cognitive Computing* 2 (11 2016). doi:10.31436/ijpcc.v2i2.39

[3] Cyprien Buron, Jean-Eudes Marvie, Gaël Guennebaud, and Xavier Granier. 2015. Dynamic on-mesh procedural generation. In *Proceedings of Graphics Interface 2015* (Halifax, Nova Scotia, Canada) *(GI 2015)*. Canadian Human-Computer Communications Society, Toronto, Ontario, Canada, 17–24. doi:10.20380/GI2015.03

[4] Daniel Michelon De Carli, Fernando Bevilacqua, Cesar Tadeu Pozzer, and Marcos Cordeiro d'Ornellas. 2011. A Survey of Procedural Content Generation Techniques Suitable to Game Development. In *2011 Brazilian Symposium on Games and Digital Entertainment*. 26–35. doi:10.1109/SBGAMES.2011.15

[5] Agata Ciekanowska, Adam Kiszczak Gliński, and Krzysztof Dziedzic. 2021. Comparative analysis of Unity and Unreal Engine efficiency in creating virtual exhibitions of 3D scanned models. *Journal of Computer Sciences Institute* 20 (Sep. 2021), 247–253. doi:10.35784/jcsi.2698

[6] Volker Claus, Hartmut Ehrig, and Grzegorz Rozenberg. 1979. *Graph-grammars and their application to computer science and biology: international workshop, Bad Honnef, October 30-November 3, 1978*. Vol. 1. Springer Science & Business Media.

[7] Hartmut Ehrig, Annegret Habel, and Hans-Jörg Kreowski. 1992. Introduction to graph grammars with applications to semantic networks. *Computers Mathematics with Applications* 23, 6 (1992), 557–572. doi:10.1016/0898-1221(92)90124-Z

[8] H. Fahmy and D. Blostein. 1992. A survey of graph grammars: theory and applications. In *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems*. 294–298. doi:10.1109/ICPR.1992.201776

[9] Jonas Freiknecht. 2021. *Procedural content generation for games*. Ph. D. Dissertation. Mannheim. https://madoc.bib.uni-mannheim.de/59000/

[10] Sverre Magnus Haakonsen, Anders Rønnquist, and Nathalie Labonnote. 2023. Fifty years of shape grammars: A systematic mapping of its application in engineering and architecture. *International Journal of Architectural Computing* 21, 1 (2023), 5–22. arXiv:https://doi.org/10.1177/14780771221089882 doi:10.1177/14780771221089882

[11] Tao Jiang, Ming Li, Bala Ravikumar, and Kenneth W Regan. 2009. Formal grammars and languages. In *Algorithms and Theory of Computation Handbook, Volume 1*. Chapman and Hall/CRC, 549–574.

[12] Tomáš Kalva and Vojtěch Cerný. 2023. Shape Grammars for Level Design of Action Role-playing Games. In *2023 IEEE Conference on Games (CoG)*. 1–8. doi:10.1109/CoG57401.2023.10333242

[13] Lila Kari, Grzegorz Rozenberg, and Arto Salomaa. 2013. L systems. In *Handbook of Formal Languages: Volume 1 Word, Language, Grammar*. Springer, 253–328.

[14] George Kelly and Hugh McCabe. 2006. A Survey of Procedural Techniques for City Generation. 14 (01 2006). doi:10.21427/D76M9P

[15] Ole Kniemeyer, Gerhard H. Buck-Sorlin, and Winfried Kurth. 2004. A Graph Grammar Approach to Artificial Life. *Artificial Life* 10, 4 (2004), 413–431. doi:10.1162/1064546041766451

[16] Stephane Lavirotte and Loic Pottier. 1998. Mathematical formula recognition using graph grammar. In *Document Recognition V*, Daniel P. Lopresti and Jiangying Zhou (Eds.), Vol. 3305. International Society for Optics and Photonics, SPIE, 44 – 52. doi:10.1117/12.304644

[17] Yuanyuan Li, Fan Bao, Eugene Zhang, Yoshihiro Kobayashi, and Peter Wonka. 2011. Geometry Synthesis on Surfaces Using Field-Guided Shape Grammars. *IEEE Transactions on Visualization and Computer Graphics* 17, 2 (2011), 231–243. doi:10.1109/TVCG.2010.36

[18] Paul Merrell. 2023. Example-Based Procedural Modeling Using Graph Grammars. *ACM Trans. Graph.* 42, 4, Article 60 (July 2023), 16 pages. doi:10.1145/3592119

[19] Kathryn E. Merrick, Amitay Isaacs, Michael Barlow, and Ning Gu. 2013. A shape grammar approach to computational creativity and procedural content generation in massively multiplayer online role playing games. *Entertainment Computing* 4, 2 (2013), 115–130. doi:10.1016/j.entcom.2012.09.006

[20] Mine Özkar and Sotirios Kotsopoulos. 2008. Introduction to shape grammars. In *ACM SIGGRAPH 2008 Classes* (Los Angeles, California) *(SIGGRAPH '08)*. Association for Computing Machinery, New York, NY, USA, Article 36, 175 pages. doi:10.1145/1401132.1401182

[21] Érica Pinheiro, Gelly Mendes, Doris Kowaltowski, and Gabriela Celani. 2007. A DESIGN TEACHING METHOD USING SHAPE GRAMMARS Regiane Pupo. https://api.semanticscholar.org/CorpusID:38103711

[22] Przemyslaw Prusinkiewicz. 1986. Graphical applications of L-systems. In *Proceedings of graphics interface*, Vol. 86. 247–253.

[23] Przemyslaw Prusinkiewicz1/2, Jim Hanan3/4, Mark Hammel1/2, and Radomir Mech1/2. 1996. L-systems: from the theory to visual models of plants. (1996).

[24] Tanya Short and Tarn Adams. 2017. *Procedural generation in game design.* CRC Press.

[25] Mark Steedman. 2003. Formal grammars for computational musical analysis. *INFORMS Atlanta October* (2003).

[26] George Stiny and James Gips. 1971. 'Shape Grammars and the Generative Specification of Painting and Sculpture'. *IFIP Congress* 71, 1460–1465.

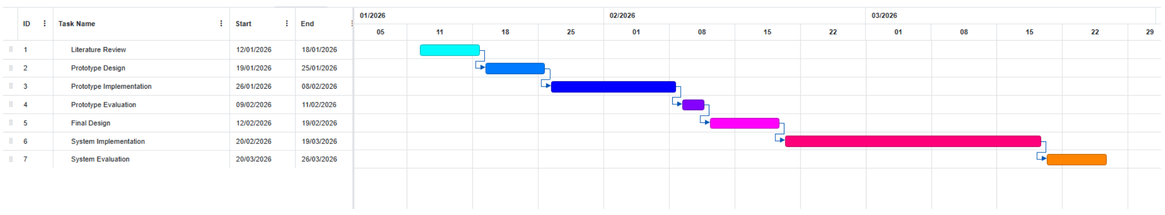
[27] Arie Vatresia, Ferzha Putra Utama, and Adi Yulianto. 2023. DESIGNING A 3D ROGUELIKE GAME WITH PROCEDURAL CONTENT GENERATION USING THE GRAPH GRAMMARS METHOD. *Jurnal Teknik Informatika (Jutif)* 4, 6 (2023), 1437–1446.

[28] Glynn Winskel. 1993. *The formal semantics of programming languages: an introduction.* MIT press.

[29] Manuel Zechmann and Helmut Hlavacs. 2025. Comparative Analysis of Procedural Planet Generators. arXiv:2510.24764 [cs.GR] https://arxiv.org/abs/2510.24764

A Project Management

A.1 Gantt Chart



ID	Task Name	Start	End
1	Literature Review	12/01/2026	18/01/2026
2	Prototype Design	19/01/2026	25/01/2026
3	Prototype Implementation	26/01/2026	08/02/2026
4	Prototype Evaluation	09/02/2026	11/02/2026
5	Final Design	12/02/2026	19/02/2026
6	System Implementation	20/02/2026	19/03/2026
7	System Evaluation	20/03/2026	26/03/2026

Fig. 7. Gantt chart for the project

B Risks and Mitigation

This section will run us through the main risks of the project - Scope Creep and 3D asset use.

B.1 Scope Creep

While additions to the projects can be useful, the danger of adding too many features is a problem in any project - it can lead to missed goals, missed deadlines, conflicts with the key requirements and other issues. This is called scope creep. In this project the main feature where Scope Creep can become an issue is the adjustable properties for the generation as there is not a set number of them required to meet the project goals and realistically a very large amount of them could be added.

For the purpose of mitigating this, the project will aim to clearly define the number and type of the properties before implementation is made and stick to those very closely. After the 'Final Design' task no more properties will be added to the system. Additionally, during prototype evaluation, only properties that were found to be feasible for the project be considered as well as only a limited amount of them.

B.2 3D Asset Use

Procedural generation requires a pre-made model as a foundation for the generation - this can then be formed in a way which is required. In this project, the model will be made by hand if possible, however it will be made only after the program is developed enough to be able to say how it needs to look. Because of that it's possible that it may take too long to create the model, or it may be difficult to have it adapt to the program.

To mitigate this, if a model is not made in time to work for the prototype, a pre-made 3D asset will be used.

C Generative AI

Generative AI is not planned to be used in this project in any capacity.

D Professional, Legal, Ethical and Societal Issues

Because of the fact that this project does not use a focus group as an evaluation method and does not really have a capacity to be misused, the ethical issues are only related to those could be made during development.

If pre-made assets are used, its copyright license has to be respected.

As well as that, any piece of software should follow a code of ethics in order to stay professional and ensure a responsible development process. For this reason I will follow the BCS Code of Conduct as I work on this project.