University of Sheffield

Imperfect Plants

Zuhaira N. Zakaria

Supervisor: Dr. Kirill Bogdanov

This report is submitted in partial fulfillment of the requirement for the degree of BSc in Computer Science by Zuhaira N. Zakaria.

in the

Department of Computer Science

May 8, 2024

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Zuhaira N. Zakaria

Signature: Zuhaira N. Zakaria

Date: May 8, 2024

Abstract

As gardening gains popularity, there is a growing need for innovative tools that empower gardeners to plan and choose the right plant in the right environments while optimizing their space. This report presents the development of a garden-planning 3D simulation tool tailored for gardeners, aiming to enhance their garden appearance based on plant choices and seasonal changes to allow them to plan and take the right action.

The simulation tool allows gardeners to virtually plan and visualize their garden layout over a period, considering factors such as seasonal changing temperature, surrounding effect, and space between other plants. As a key methodological application, the study explores and applies the L-system as a generation tool for the plant assets within the simulation, known for their ability to model complex and realistic plant structures, contributing to the authenticity of the virtual gardening experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The art of garden planning offers a strategic approach for planners to decide and achieve vibrant, visually interesting and maintainable garden throughout all four seasons. Each season has its uniqueness and thus, the challenge lies in maximizing the opportunities to take advantage of nature's cycles. Gail [19] writes that to create a beautiful garden, one must observe, plan and maintain it carefully and demonstrates the significance of these aspects in achieving a stunning garden that reflects the changing seasons, available space and soil conditions throughout the year. Due to the amount of information required, this report must focus solely on garden planning. This report examines how simulations transform gardens into flexible spaces for planners by considering plant growth, interactions and seasonal effects.

As traditional garden planning was once confined to hand-drawn and trial-and-error procedures, the integration of advanced 3D simulation technologies into gardening practices opens a new perspective to visualize, plan and optimize their garden. As the report delves into its development, we will uncover one known methodological application by incorporating the L-system, a string rewriting system, that was first introduced by a biologist to illustrate neighbourhood relationships among plant cells. Developers have used it for various applications, including plant modelling and more complex production rules consisting of strings to create branching and tree structures.

L-systems are particularly adept at representing complex branching structures. Other considerations of modelling methods, such as Bézier curves, provide explicit control focusing on the precision of curves. The application of Perlin noise focuses on mimicking the irregularities in nature.

The L-system is a highly detailed method of representing plant growth. Other methods, like photogrammetry, have limitations in displaying 3D objects due to incorporating the actual plant, which makes it less procedural. The choice of modelling method depends on the realism, control, and specific project requirements. Photogrammetry is preferable for realistic visualization, but for algorithmic growth and procedural generation, the L-system is more suitable, like in this project.

The planning of a garden and the growth of its plants are closely related to how the garden will look at different times of the year. Each plant in a garden goes through its life

cycle, and this affects how the garden looks and appears. To create and maintain a certain look, gardeners need to consider managing each plant, how long a plant can survive, and what actions they need to take to ensure a structured process. This is when simulations and garden modeling come in handy. They provide a real look at chosen plants and show a detailed lifecycle to estimate all factors before planting.

1.1 Aims and Objectives

The main aim of this project is to develop a tool for planning gardens and to explore the potential of L-systems as a method for achieving this. Specifically, we will investigate how realistic plant simulations can be used to visualize garden layouts and inform planning decisions. As we look into various developments of FSPM, or 'functional-structural plant models'[12] have continuously been beneficial by providing a platform for systemic studies, improving understanding and conveying meanings of how complex plant systems work. As a result, the knowledge and models have huge potential in applied plant sciences where they assist in the refinement of agricultural, horticultural and forestry practices [8]. Those FSPMs had always been developed in different models to solve different problems facing various practical agricultural issues. Among those FSPM, the L-system, introduced by Aristid Lindenmayer [14], where the strings produced are interpreted as turtle commands, has captivated many programmers who use it to model complex plant morphogenesis and obtain plant-like geometric shapes. From the L-system, many different fractals can be created by exploring different rules and combinations.

Throughout this research, a garden planning simulation will be developed. The garden planner will allow users to select plants, specify their type and track their growth on a weekly timeline. The L-system will generate tree shapes based on the growth rules of each plant. Users will also be able to interact with the environment and observe how the scenery changes throughout the four seasons.

1.2 Overview of the Report

The next *chapter 2* will cover a *"literature survey"* that starts by explaining the concept of the L-system generally and how it's suitable to be adapted into garden planning simulation. Further topics such as Plant interaction will be discussed more in different methods since the L-system is only focusing on growth. In between, since the author's degree is not closely related to nature, biology and plants, a simpler data structure, idea and implementation of seasonal changes, time and space of plant growth will be discussed further.

In *chapter 3*, *"Requirement and analysis"*, a general analysis of the problem is presented as well as listing the requirements and methods of their evaluation.

In *chapter 4*, for this stage, a *"Progress"* section is added to discuss some progress of the first semester throughout this research.

Chapter 2

Literature Survey

Various dynamic modelling approaches have been introduced to derive models of plant growth. Simulation can aid in solving issues related to gardening. This work has been dedicated to applying visually explicit plant models, which are constantly improving from 2D to 3D architecture to accommodate specific needs. Among those, the most influential work was done by Liedenmayer[14] [12]. All visual plant models used in this work can also be called *FSPM*, functional-structural plant models and one way to describe plant structure and dynamics is through intuitive rules like the L-system.

2.1 What is Garden Planning Simulation Tool?

A simulation can be defined as the imitation of a situation or process. In this case, simulating a garden provides a broad overview of the challenges faced by gardeners when planning their gardens. For this report, the main focus is on how gardening tools can generate the appearance of a garden during a specific time period. In addition, the individual can effectively plan and manage appropriate plants for a specific period, as well as ensure successful management for the following season.

2.2 Previous work

There are several plant simulations available, and some of them require purchase. "TreeIt" is one such simulator, which allows for the creation of realistic-looking plant assets that can be used by game developers or simulators. Another simulator, called "The Grove," was introduced in 2014 and focuses on simulating the growth of plants in a forest environment. It comes with tools for pruning, drawing, reacting, and bending. In addition, there are many garden planning software applications available that allow users to create garden designs in 2D and transform them into 3D views. One such paid software is "Garden Planner." However, there are not many simulators that focus on plant growth in a garden setting over some time.

Under the topic 'Imperfect plant', three individuals explored the L-system and applied it in various ways. One of the works written by Yuncheng Wang [21] aimed to create a garden

simulation by utilizing real-time growth data, considering factors like sunlight as fundamental components for their calculations. It also features Bresenham's line algorithm [7] applied in the interaction between plants and the branching system by calculating space voxels and recording whether certain voxels have been occupied or otherwise. The writer also wrote one of the significant highlights that could be related to this work further is the effect of season on a plant's structure but didn't apply it within their works due to different aims.

Another work by Hubert Krawczyk [11] has shown significant improvement related to previous work by adding more calculations of real-time growth data considering two factors, which are water and sunlight. More detailed tests were shown and also applied to Bresenham.

2.3 Seasonal Highlights

Anyone would love to see vibrant flowers, strong trees, and calling landscapes wherever they go. It's worth mentioning the impact of extreme weather events, such as floods, on types of plants in small gardens in familiar locations due to future climate changes. As extreme weather conditions require more time, resources, and deep calculation, it results in the need for alternative measures to simulate these changes. This report will target small gardens in familiar locations, focusing on simple solutions to address four seasons as a basic climate change. Climate change does have a significant impact on gardens across the UK, influencing their design and character. The character or design of a garden has always been associated with climate change, as mentioned in [9].

2.3.1 Factor affecting garden characters

Climate changes are dependent on regional locations. For example, western parts of the United Kingdom, have historically had higher rainfall patterns, and continuously have severe winter storms that will increase the possibility of floods, such as in 2021 [3]. Some locations such as London also show an incidence of drought stress due to lower summer precipitation and higher rates of soil moisture evaporation. Good planning decisions are necessary for areas affected by constant climate changes. These are some of the changes that could occur in a small garden throughout the year. By limiting the scope to the four seasons, we assume a more stable climate throughout the year at a certain location for the garden simulation.

The document states that the character of a garden is influenced not only by its location, but also by personal preference, primary use, lifestyle, and biotic factors. Gardens can range from the highly functional to the highly aesthetic [9]. Depending on the planner and their reasons for planting a garden, the needs and choices for garden simulations may vary. This can result in the selection of different plants for the visualization of the garden. An article by Mark Wolfie [20] expressed that the best result of a beautiful landscape can be achieved by researching and organizing flower beds, trees, and shrubs while mixing them with perennials that bloom at strategic times. Designing a garden planning simulation allows users to choose garden plants and be able to see the growth for a long period to find the best landscape outcome before actually planting them, significantly improving the planning process as one

of the benefits of plant models stated by ALS which is a garden, landscaping and driveways experts company [4].

2.3.2 Choosing the right plant

Different individuals will have different ideas or choices of plants they want to look into. For example, Mark Wolfie [20] mentioned that in mild climates, most gardeners would like to have 'vinca', 'begonia', and 'impatiens' in summer. For the cool season, pansies and snapdragons in winter. Other than based on appearance, article [15] shows choices of plants in groups of lifecycles and specific characteristics:

Plant Types	Life Cycles Duration
Annuals	one year
Biennials	two years
Perennials	three years or more

Table 2.1: Plant Life Cycle Types

Plant Categories	Descriptions
Shrubs	Woody branches and no trunk
Trees	Larger than shrubs and have a trunk
Climbers	Grow upward
Bulbs	Underground storage organs

Table 2.2: Types of Plants and Their Descriptions

Bedding [15] refers to plants that are planted temporarily in flower beds or borders. This could include several of the plants from the groups, such as half-hardy annuals or tender perennials. Some are bulbs or shrubs. The upcoming section will discuss more detailed information on plant structure and branch complexity, as different simulation levels may be required depending on the plant type.

2.4 Time and space to grow

Depending on the plant chosen, each plant requires optimal space and time to continuously strive in its lifecycle. From a common point of view, we can see several phases that plants go through during their lifecycle. For example, sunflowers start from seed germination, become seedlings, and turn into vegetative form until the bud stage is achieved, where buds begin to show at the top of the stem. Later, flowering and ripening stages come along. Some plants could die without proper space, environment, soil condition, and water. There were very diverse solutions for common problems in developing garden simulations. Some of the extreme conditions such as plant disease, the effect of typical genetic anomalies, and extreme weather changes not covered in this work. Although the limitation is the same as [17], the difference will be applying seasonal changes to see how the garden will look.

In this research, we will only focus on the physical appearance of plants, rather than exploring all the granular components that affect plant growth, such as the amount of sunlight, water, and other nutrients. While granular components are important for the growth and health of plants, this research will assume that they are already perfectly cared for. Instead, the focus will be on the survival of plants during seasonal changes and their lifespan aspects.

2.4.1 Growth cycle

The lifecycle of trees varies depending on their grouping and the length of time they can survive through different seasons. One model, created by [2], can limit their tree generation to have single, non-hollow trunks. Since it's proven to be able to generate plants by finding the rules and structures of plants, it'll also be possible to create a 3D simulation of plant growth for small garden plants. Paper [17] encode biological observation to simulate the structure growth of non-woody plants using the L-system. This is then further improved by other works that focus on specific needs such as simulating a plant attacked by insects and the effect of pruning on a plant's structure.

The focus of the above works has been on instant fixed plant asset generation. In this report, a major difference in adding seasonal changes requires a parameter for this timeframe that must be considered to use the correct range of period growth in visualization. The weekly growth rate is chosen. This is because weekly growth has rapid changes, making it easy to assume optimal growth until certain conditions are not met. These conditions are related to the life cycle of the garden plants' needs such as watering, sunlight, and soil condition.

2.5 Plants' interaction

In scientific terms, plant interaction has always been referring to the interaction with other organisms. In nature, some main common interactions are competition, symbiosis, mutualism, parasitism, and secondary metabolites [5]. A simulation [17] can encode using an extension of the L-system rules together with insects travelling through branches to simulate the parasitism interactions. This model can be included in simulation educational purposes to control weeds and observe the impact of insects on crop plants [18]. Due to the real-time growth rate, and interactions between plants and the environment from [17], Garden Planning, we will prioritize the aesthetics of the garden. As we focused on the aesthetic of the garden, it's also worth pointing out that plants not only adapt to photosynthesis and transpiration but also respond structurally to environmental stimuli. [12]. With the help of simulation garden planning, it is possible to generate a viable plant structure design for the growth of plants.

The primary factor that affects plant growth is the interaction between plants. It is important to ensure that plants grow in an organized manner, without any overlap between their growth patterns. To achieve this, using Bresenham's line drawing algorithm to visualize between-plant interactions can be a helpful tool. This technique can help in establishing better management strategies for gardens and improve the randomness in plant structures.

2.5.1 Bresenham's line drawing algorithm

Bresenham's line algorithm is a computer graphics technique used to draw lines in a 2D coordinate system, which can also be extended to 3D. It is an efficient and simple method for determining the connecting points between two given points and drawing the line between them.

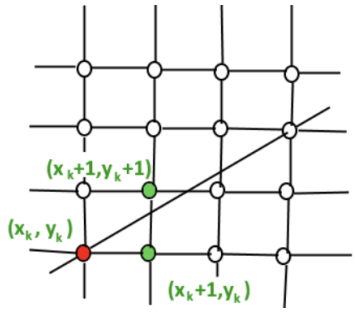

Figure 2.1: 2D coordinate system, showing points to choose on specific line [1]

Given two 2D coordinates, the required output is all the points on the line connecting them, as shown in Figure 2.1. To decide which point to pick next, we need a decision parameter. The parameter should keep track of the slope error from the previous increment to y. If the slope error becomes greater than 0.5, it means that the line has moved upwards by one pixel. We must then increment our y coordinate and adjust the error to represent the distance from the top of the new pixel. To do so, we subtract one from the error.

In a 3D simulation, a plant collision occurs when multiple plants occupy the same space, which is not logical. To simulate the growth of plants realistically and enable them to interact with one another, a mechanism that detects the presence of other branches is required. Three steps will be followed to achieve this. To begin with, each of the new branches will be categorized according to the position of its cells. The next step involves evaluating the presence of any existing branches before allowing a new branch to grow in a particular direction. In case there are other branches, their growth direction will be adjusted accordingly. Lastly, the dynamic adjustment process will have more detailed rules to avoid any potential collisions during the adjustment.

2.6 L-system

Over the past 20 years, a variety of implementations of L-systems [14] have been designed. The most highlighted plant modelling has been *cpfg* a program for simulating and visualizing plant development, based on the theory of L-systems [17]. The research shows the use of mathematical notation based on formal language theory to write the L-system rules and extend them into C-like statements for specifying the parameter changes. Later gave rise to the new program *Ipfg* and modelling language L+C. Another implementation of a plant modelling system related to the L-system is XL [10][6] which can manipulate much

more dynamic structures made of modules. It is possible to define production rules on structures such as graphs, rather than just on plants, impacting the extension L-system [6]. Furthermore, [6] explored and developed the L-system as a comparison between dynamic and static languages. Paper [6] suggested that the use of dynamic languages is particularly well adapted to the building of simulation systems in developmental biology, since dynamic languages can appear at first sight to be technical in nature and less interesting to biology. Despite various approach and languages used by the developer, the L-system ability to describe models of plant development [17] have been a major focus.

2.6.1 The generation mechanism

L-system is composed of 4 components by labelling them as V, ω, P, and development *Mechanism*:

1. V refers to strings of characters. Each character in turn consists of symbols that can be of two types: variables, which can be replaced, and constants, which are not replaceable.
2. ω, refers to the initial axiom string. This is where the construction can start.
3. P refers to 'Production rules', that expand each symbol into a larger string of symbols.
4. *Mechanism* is the process of translating the generated strings into geometrical structures. This is where many developers could explore various ways of applying the L-system.

An example of simple production rules can be shown below:

- Let Production rule be: 'F $\rightarrow$ 'FF
- Let axiom be: 'F

n = 0:	F
n = 1:	FF
n = 2:	FFFF
.	.
.	.
.	.
n = *m*:	2^m of F

Figure 2.2: Example of basic 'string replacement' of L-system

We previously saw a simple string replacement, changing ‘F to ‘FF, which can be used to trigger certain visualization actions such as moving forward. The action to move forward will be translated by mechanisms supporting the translation between the string and actions. Further improvement involves creating a mechanism that translates a string into more complex actions or visuals, such as going left or right. Assuming a simple L-system mechanism that moves forward(F), left(-, 90 degrees), and right(+, 90 degrees). To proceed with the next step, we need to provide a production rule similar to the one used in the previous example.

- Let Production rule be [17]: ‘F → ‘F-F+F
- Let axiom be: ‘F

This rule state that whenever there is a ’Forward’ command, it will be replaced by a series of commands which include turning left and right. A sequence of results for n ranging from n = *1,2,3,4* is shown in Figure 2.3.

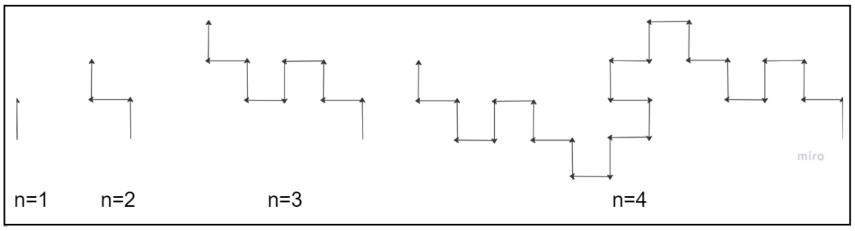

Figure 2.3: result of *n=1,2,3* of ‘F,-,+ drawn

Other research has explored various production rules, such as the construction rule depicted in Figure 2.4, which is capable of generating fractals to visualise a 2D plant-like structure such as Figure 2.4. The next section will discuss further how the idea of production rules can help the research to understand the rules of future plants to be designed.

Figure 2.4: enhanced plant from [16]

2.6.2 Production rules

According to paper [17], modeling plants requires further refinement of the L-system and programming language to facilitate the application of the system and generate the desired plant model. A decision framework to develop certain rules of plants is mentioned here by adapting and researching basic mechanisms that control plant development: lineage, capturing the class of context-free L-systems, and endogenous interaction, captured by context-sensitive L-systems [17]. Then able to show how each development of flower, branch, and cell division during the growth process of plants.

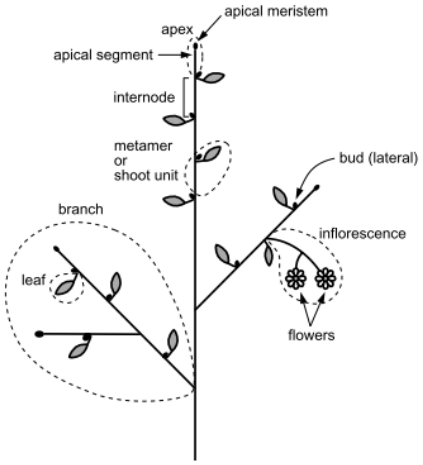

Figure 2.5: Example of selecting module, possible fractals

To elaborate on these ideas, it is important to monitor the growth patterns of individual plants and modify the production rules accordingly. By adjusting parameters such as angle, length, and branching probability, we can fine-tune the process to create a more diverse and realistic range of plants. Let's examine Figure 2.4 in detail. The main stem repeats itself, giving rise to a fern-like structure that branches out at 45-degree angles to the left and right. Additionally, one more stem is added to the main branch, resulting in a well-defined pattern. This demonstrates how we can identify the structure of a plant and convert it into a set of production rules.

Furthermore, the research enhances a more realistic order of generation rule illustrated in Figure 2.6. The figure shows how a node with a thicker line or different age-like values will generate the stem, while the other node with a thinner line will branch, depicting realistic growth branches that will be younger on top and older below, which will not branch as it gets older be used to model the likeness of a larger group of plants such as Maple trees and more.

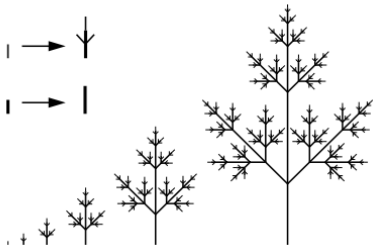

Figure 2.6: Internode idea, image by [17]

2.7 Technologies

One of the key components of an L-system is its mechanisms for modeling plant growth. To apply the L-system, it is essential to consider two important technologies: programming language and software for the development environment. Some aspects that can be compared between software programs that are commonly used are Unreal, Unity, Blender, and OpenGL, are as follows:

1. Familiarity with programming language
2. 3D graphics capabilities
3. Ease of Prototyping and experimentation

2.7.1 Familiarity with programming language

Understanding software time consumption can be reduced by being familiar with the programming languages used. The programming languages that can be applied by those software programs are as follows:

1. Unreal : C++ and Blueprints
2. Unity : C#
3. Blender : Python
4. OpenGL : Java and C++

During my studies, I found that Java and Python were the most commonly used languages, while C# was used minimally during workshops. Although I haven't fully explored C++, I believe it can be understood with some time and effort. In my experience, Python is the most preferable language due to its easy-to-use libraries. This allows me to focus on data handling and algorithms instead of spending time on understanding the language as a whole.

2.7.2 3D Graphics Capabilities

When comparing the 3D graphics capabilities of different tools, each platform offers unique strengths. Unreal Engine is known for delivering highly realistic visuals due to its advanced lighting and shading features, making it excellent for creating intricate, lifelike environments.

Unity supports both 2D and 3D rendering, giving developers the flexibility to choose their desired art style and work with pre-configured rendering systems.

Blender provides an extensive suite of modeling and rendering tools, such as the Cycles and Eevee render engines, which enable users to produce high-quality animations and visual effects. OpenGL, while powerful in terms of customization, requires significant programming knowledge and can be challenging for beginners.

Overall, each tool provides different levels of sophistication and usability, catering to a range of user needs and project requirements. Each software hasn't been explored beforehand, so time is needed to learn. Based on the aim of creating seasonal changes, Blender shows the most achievable capabilities due to its ready-made visual effects. Overall, each tool provides different levels of sophistication and usability, catering to a range of user needs and project requirements.

2.7.3 Ease of prototyping and experimentation

When it comes to prototyping and experimentation, each software has unique characteristics that impact how quickly and effectively new models and ideas can be developed. Unreal Engine allows users to rapidly test ideas through its visual scripting system called Blueprints, which eliminates the need for in-depth coding knowledge.

Unity also facilitates experimentation with its asset store, where users can find a wide array of pre-made resources, and with tools like ProBuilder, which simplifies level design.

Blender provides non-destructive modeling tools like modifiers, enabling quick adjustments without affecting the original model, and its scripting features allow for automating repetitive tasks.

Meanwhile, OpenGL is more challenging for rapid prototyping because it requires significant programming expertise to manually implement graphical elements. Overall, Blender and Unreal stand out for their user-friendly features that make them suitable for testing and refining creative concepts efficiently.

2.7.4 Scoring table

Aspect	Unreal	Unity	Blender	OpenGL
Familiarity of programming language	3	4	5	2
3D Graphics Capabilities	5	4	4	3
Ease of Prototyping and Experimentation	4	4	5	2
Total	12	12	14	7

Table 2.3: Adjusted Comparison of Software Programs

The scoring of each software program was adjusted to reflect their specific strengths and weaknesses. Unreal Engine scores 3 in programming familiarity because it requires C++ or the simpler but still somewhat complex Blueprints. Unity scores 4 due to C# being more approachable. Blender scores 5 for using Python, known for being beginner-friendly, while OpenGL scores 2 as it requires expertise in Java or C++. For 3D graphics capabilities, Unreal leads with 5 due to its advanced, realistic graphics, Unity scores 4 for offering HDRP, Blender scores 4 for versatile modeling and animation tools, and OpenGL scores 3 due to its reliance on manual implementation. Lastly, in prototyping and experimentation, Unreal and Unity each score 4 for their built-in tools and templates, Blender scores 5 for non-destructive modeling and scripting, and OpenGL scores 2 for its complex, slower prototyping process.

Python, particularly Blender's Python API, has been identified as the most suitable choice for Garden Planning. However, if the performance of the system is a major concern, C# is considered a better option. For Garden Planning, Python will be used to develop the simulation of Garden Planning, applying L-system as the main method.

Several works have used Python for plant's structure in 3D environment, such as L-Py [6]. It presents an adaptation of L-systems to the Python language, demonstrating the use of dynamic language properties, which enhances the development of plant growth models. In another bachelor's thesis, the L-Py framework has been applied as an add-on for Blender with a user-friendly interface, enabling various capabilities in plant changes, such as shedding of branches, pruning, soil nutrient distribution, and more [13]. Various software for development, such as Unity, Unreal, and Blender, can be chosen due to their game-engine capabilities. In previous work related to "Imperfect Plant", a full 3D implementation has been achieved using a Python environment by importing turtle.

In a paper by [6], different programming languages were discussed for early exploration of L-system. These included L + C, which is an extension of C++ that focuses on L-system production, and XL, which has Java as a support language. Each of these languages has its own strengths for L-system production programming. For instance, dynamic languages like Python allow for more flexibility in variable specifications, while statically-typed languages like C# and Java prioritize the optimization of data structure handling and computation efficiency.

2.8 Summary

Garden planning has various benefits for gardeners. It allows them to design, visualize, observe the growth virtually and analyse any management of the garden ahead of actually planting. Various methods and software have been introduced to develop elaborate functional-structural plant models to be used in problem-solving software. As discussed in this report, L-system has shown impressive values for creating iterative and controllable plant structures. Despite not having any support for interactions, the additional method of Bresenham's line has also given a good solution for a dynamic garden. Additionally, choosing the right development mechanism is crucial for feasible and usable specifications that align with project requirements.

Chapter 3

Requirements and Analysis

3.1 Motivation

In Chapter 1, the project focuses on simulating the growth of plants through seasonal changes. The objective is to observe the changes in plants over time. Users will have the ability to select various types of plants and set timeframes containing seasonal changes, creating a small garden simulation. Ideally, users should also be able to interact with the simulation. Most garden planning software available today only allows for static garden arrangement planning and doesn't simulate plant growth. As a result, it's hard to visualize the climate and see dynamic seasonal changes in garden planning. This project aims to provide a user-friendly platform for visualizing seasonal changes and observing how a garden will look after a certain period.

3.2 Analysis

To develop a dynamic Garden Planning simulation, it is crucial to consider the most suitable method for generating the structure of each plant. Ideally, this method should be controllable so that it can be adjusted based on the growth rate and the scene timeline which will relate to the seasonal changes accordingly. The L-system method has been introduced to significantly assist plant structure development. With the addition of Bresenham's line algorithm, it supports the interaction between plants and surroundings and a well-elaborated garden can be generated. Apart from that, the choice of the development system has also been crucial for the accessibility for users to use the simulation at ease. As a result, providing a well-structured and dynamic program that meets all requirements is a useful way to assist gardeners in observing plant growth throughout the seasons.

Users can interact with plants in the simulation by selecting and placing them in a location, and adjusting parameters such as the initial timeframe. Some plants may be older and require several phases to simulate already-grown plants or to be moved from pots to soil. The location of the plant affects its appearance and behavior. Once the user has set up the environment, they can start the simulation. To accurately model the growth of plants,

a time control is necessary to show changes each week until the specified timeframe, rather than only the final result. This will provide a more realistic representation of plant growth.

In Chapter 1, production rules should be designed reasonably according to certain types of chosen plants to show the least amount of unique details for the user to recognize the plants and their growth phases. The seasonal parameters should be possible to modify, so users can make more adjustments for some variety of seasons. However, for minimal achievement in this research, a four-season simulation will be adapted as the default environment of the gardening location.

3.2.1 Plant's growth into garden planning simulation

The production rule extension groups plant structure and components into three levels of complexity based on theoretical growth. The L-system is used as the primary method for plotting the structure of the plant. It also dynamically checks the environment and creates different plant assets using the correct production rules that represent the selected plant types in the simulation. Thus, simulation's complexity is represented by three levels of complexity with five aspects to be compared:

Aspect	Low-Level	Mid-Level	High-Level
Lifecycle	Basic stages perhaps seed, sprout, mature and reproduce	Additional stages and variations such as flowering seasons, fruiting period and shedding phases	Detailed representation of plant's lifecycle including environmental factors, multiple flowering and fruiting cycles and response to external stimuli
Height	Basic rules determining plant's height with limited variation	Factors affecting height including spaces between other plants	Complex rules considering a range of environmental conditions, genetic factors and any mutations affecting height
Branching	Simple patterns with limited variability	Complex branching considering factors like top growth control and side branching	Intricate branching potentially influenced by environmental conditions, competing nearby plants and external stimuli
Objects	Basic representation of leaf, flower, fruit, or stem and minimal variation	Varied shapes, colors of those objects based on environmental conditions	Detailed variations considering genetic diversity, interaction with other plants and evolutionary changes over time
Growth Rate	Constant rate	Variable growth rates based on external factors like seasonal changes, weather or nutrient	Dynamic growth rules influenced by intricate environmental factors, age and interactions

Table 3.1: Categorized Aspects of Plant Production Rules

Table 3.1 we analyzed the specific needs of different levels of modeling. To create feasible software within the given timeframe, we can set requirements by prioritizing the minimum processes and achievements necessary to showcase the concept of gardening tools. It is possible to use the rules for a plant's growth as a low-level presentation for garden

planning. Even if two objects share similar shapes, they can still be distinguished by specific characteristics such as the color of their flowers, stem sizes, and lifespan. If the project is to progress further, more high-level rules and models can be developed to enhance user freedom and increase the variety of plant choices with improved details and capabilities.

3.3 Project Requirements

The guidelines provided in Table 3.1 can assist in analyzing the necessary requirements to develop a basic garden planning system. The objectives of the system can be determined, and the appropriate processes can be identified to enhance its capabilities. This approach allows for scalability and simulation of various aspects of plant development. To achieve this, it is essential to consider all low-level complexities and seasonal aspects of garden growth. This will require scaling certain functionalities and evaluating them for any possible flaws. Below is a table that outlines a list of functionality requirements:

No.		Requirements	Importance
1. Plant's growth	1.1	System can grow individual plant with L-system rule	Mandatory
	1.2	System can grow plants according to specific plants' rules in the library of growth rules of each plant	Mandatory
	1.3	Plant should refrain from branching when the surrounding space occupied, demonstrating plant-to-plant interaction	Desired
	1.4	Plant growth should be visualised weekly	Mandatory
2. User interaction	2.1	User can pick plant's type, age, and location	Mandatory
	2.2	User can locate multiple plants in a scene	Desired
	2.3	User can interact with the scene and plants in real-time during the simulation	Desired
	2.4	Users can define the absolute run time or pause	Desired
	2.5	User can change parameters of seasons such as temperature, the intensity of light, seasons and weather	Optional
3. Seasonal effect	3.1	The system can simulate all four seasons during runtime. The seasons are easily distinguishable from each other.	Desired
	3.2	The system accurately visualizes changes in the scene's background to seasonal variations, temperature, and light intensity parameters.	Optional

Table 3.2: Project requirement

To fully realize the potential of a seasonal garden planning tool, this project is divided into three main parts: Plant Growth, User Interaction, and Seasonal Effects. Within these three parts, there are three different levels of importance: 'Mandatory', 'Desired', and 'Optional'. The 'Mandatory' aspects are the main focus of applying the L-system method in the creation of dynamically growing plants based on runtime, mainly for individual plants. Once this system can grow plants using L-system (1.1), follows good production rule from library creation (1.2), follow weekly growth rate visualisation (1.4) and allow the user to pick initial parameters of a plant (2.4), it will be an adequate system to show growing plants in the virtual world.

However, in order to achieve garden planning, some 'Desired' aspects should be achieved, such as Interaction between plants (1.3), putting multiple plants into the scene (2.2), and running the simulation accordingly (2.4). After achieving the garden level of growing plants, seasonal aspects of the simulation will be considered to show further interaction or growth effects when changing temperature, climate and seasons. (2.5) mentions changes in parameters that have a crucial impact on the growth rate of various plants. Some extreme or desired climates by the user at a high level of complexity of development will require changes in a plant's growing pattern. For example, when a plant doesn't grow in suitable conditions, the lifecycle may terminate instantly, while changing the lifecycle or age of a plant too high may not be appropriate for early plantation of garden planning plants.

3.4 Evaluation

In this chapter, we have explored the basic components of the Garden Planning System that we have proposed. Our focus has been on the design and growth structures of plants, the selection of suitable plants for small-scale gardens, and the complexity inherent in individual plants, as well as determining the overall project scope. As we move forward with the development of a system with practical mechanisms, we envision a simple yet effective plant growth simulation. Additionally, as we progress beyond the foundational concepts of the L-system, we aim to enhance this system by incorporating multiple plants and dynamic seasonal aspects. In terms of strengths, the project aims to capture the growth structure details of plants, allowing users to recognize and choose plants based on their needs. The choice of a small-scale garden demonstrates a user-centric approach, ensuring that the project scope focuses on providing a single solution to a single problem: the ability to demonstrate the appearance of a garden after a period of time.

In conclusion, by conducting a practical comparison, considering production rules and detailed requirements, and evaluating the feasibility of the project's progress, we can ensure that we are on track to achieve our goals. Any shortcomings will be analysed and addressed to ensure a successful outcome.

3.5 Ethical, Professional and Legal Issues

This project does not involve a human aspect and will not collect any personal data. All external work or resources the report uses are referenced properly. All software used is acquired legally and is used as intended and allowed in the corresponding licence agreements.

Chapter 4

Design

4.1 Introduction

This chapter will discuss further into the implementation of the Garden Planning System, integrating insights from previous chapters on garden planning. Blender has been selected as the primary environment for this project. While L-systems have proven valuable, adjustments may be necessary as the project progresses.

The objective of this chapter is to develop a solution based on the analytical insights previously discussed, focusing on the practical application within Blender and the adaptation of L-system methodologies to meet project-specific needs. Additionally, aiming to be a user-friendly platform, the whole implementation is aiming to be an add-on for the Blender to be easily adapted to any computer's blender.

4.2 Design Methodology

Various design methodologies offer distinct approaches to project management and execution. The Waterfall Model is known for its linear and sequential progression, which does not easily accommodate changes once a phase is completed. This rigidity makes it less ideal for projects requiring flexibility. The Spiral Model, on the other hand, integrates elements of both design and prototyping in a risk-driven process, making it suitable for large, complex projects, though it often leads to increased time and complexity in management since there is a limited amount of time. Agile Development emphasizes continuous improvement and stakeholder collaboration, which is valuable when frequent input from clients or end-users is available, but less so when direct client involvement is minimal or inconsistent. For this project, having client or direct feedback from stakeholders is unsure, thus couldn't focus on Agile Development.

Finally, for this project, iterative design has shown as the most effective approach. This method combines the strengths of being adaptive to changes and continuously integrating user feedback, without the overhead of the Spiral model or the intensive client involvement required by Agile methodologies. Iterative Design's cycle of planning, designing, implementing, and evaluating allows for flexibility and responsiveness, crucial for a project that integrates technical and creative aspects such as 3D modelling and Python scripting. Each iteration provides an opportunity to refine and enhance the system based testing, ensuring the final product is both user-centric and robust even in the absence of direct user feedback. Specifically in current condition, the project utilizes detailed internal testing protocols designed to simulate a range of user interactions and scenarios. This approach allows the developer to act as 'proxy users,' identifying usability challenges, functional limitations, and potential enhancements from the perspective of end-users. Internal testing within an iterative design framework offers a practical alternative that still adheres to the iterative principles of continuous evaluation and improvement. This approach ensures that the development process remains dynamic and responsive to potential user needs and preferences, ultimately leading to a robust and user-friendly product.

4.3 Detailed Design Process

In the analysis phase, several key elements were identified that are crucial for the successful development of the Garden Planning System. For the design phase, the following subsets have been selected based on their critical role in achieving the project's objectives:

1. Plant Structure Generation

    - This element was highlighted in the analysis, emphasizing the simulation's main functionality of plant growth at specific rates, aligned with the scene's timeline. This enables users to manipulate the scenery using predefined plant shapes and structures, based solely on the type of plants, their positioning, and seasonal variations. Rather than requiring users to manually create each asset and model, the system automates the entire development stage, streamlining the process for the user.

2. Interactive Plant Manipulation

    - From the analysis, it became clear that user interaction with plants—such as selecting, placing, and adjusting growth parameters—is essential. This functionality allows users to simulate different developmental stages of plants, accommodating scenarios where plants are moved from pots to soil, or where they are introduced at different maturity stages.

3. Plant Production Rules

    - The system will incorporate a set of predefined rules that dictate how plants grow, respond to environmental changes, and interact with other plants. These rules are foundational for simulating a realistic and scientifically accurate garden ecosystem. However, despite detail explanation of L-system, some adjustment has to be made due to incorporating into Blender.

4. Seasonal Changes Simulation

    - Integrating the ability to simulate seasonal changes affects plant behavior and visual appearance, offering users a powerful tool for long-term planning and management of their gardens.

These elements were selected as they directly address the user needs to be identified during the analysis phase—particularly the need for flexibility, realism, and interactive simulation capabilities in garden planning. Their development is expected to provide significant value to the users, enhancing both the usability and functionality of the Garden Planning System.

4.3.1 Design Process

These are the flow of the project, showing the design process and how each of them will be implemented:

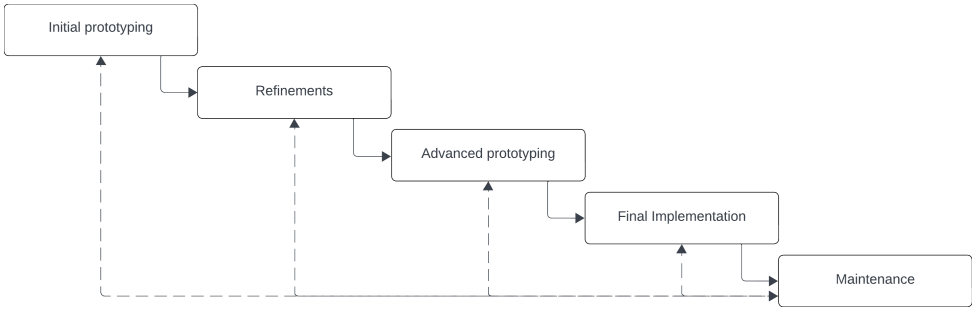

Figure 4.1: Waterfall Iterative Process

1. **Initial prototyping**: The first prototype focused on establishing a basic framework for plant structure generation and interactive manipulation. Early designs used simple geometric models to represent different plant types.

2. **Integration of Feedback and Refinements**: Subsequent iterations incorporated feedback from initial testing, leading to the refinement of the plant manipulation interface and the enhancement of the plant production rules to include more detailed biological parameters.

3. **Advanced Prototyping**: Further development introduced the simulation of seasonal changes, which required integrating meteorological data into the system. This phase also expanded the complexity of the plant models to include more detailed textures and growth patterns reflective of seasonal influences.

4. **Final Design Adjustments**: The final design phase involved optimizing the user interface for easier navigation and control, ensuring that users could effortlessly interact with the simulation features and access detailed plant data.

5. **Maintenance**: After the final release, the system requires regular maintenance to ensure continued performance and accuracy. This involves monitoring user feedback, fixing bugs, updating growth data, and enhancing the plant growth algorithms. Maintenance activities also include upgrading dependencies to the latest versions, implementing security patches, and improving performance based on emerging best practices. Additionally, continued refinement of the user interface ensures it remains intuitive for new users and supports evolving workflows. If full development coverage needs to be redone, the project can restart the process, gather feedback, and return to maintenance as needed.

As stages are completed, the project will continue iteratively from the final stage to maintenance. Additionally, there is a possibility of going back to any level to make further improvements. Throughout the process of prototyping and iteration process, finalisation is where completing the design specifications to a degree allows full-scale production or development. Depending on current skills and adaptations, the time required for full management capabilities will vary. Due to the need to adapt to new skills and time constraints, completing a full-scale development project may be challenging.

4.4 Experimental Design

4.4.1 Progress in Initial Prototyping

Firstly, a simple Python code drawn in 2D, where the circle gradually grows in the scene, depicted a view of a plant growing in size from above. It was aimed to visualised simple systems that will be needed in 3D.

The initial version of Python 2D in figure 4.2 only allows for a circle that represents the top view of a garden. However, it doesn't take into account any details about the plant structure or actual growth rate rules. Instead, it only makes assumptions. For instance, the largest circle with small dots represents the small flowers of a large tree that grows slower than others. White dots appearing in certain areas indicate flowers such as tulips, which bloom every 3 seconds, and bushes such as roses, which bloom larger flowers and grow faster than a large tree. However, these growth rules cannot be correct but can be used for user interface analysis.

The three distinct types of potential plants are demonstrated in 4.2, each varying in initial sizes and growth rates. From this process, several aspects have been analysed of the process

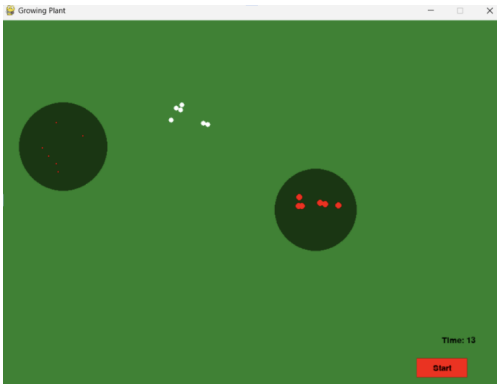

Figure 4.2: Simple Python garden planning

for implementation in Blender, including:

1. Plant Placement Strategies
2. Choosing Plant Varieties
3. Implementing Diverse Growth Protocols
4. Real-time Updates and Modifications
5. Utilization of a Dynamic Animation System

Table 3.2 shows how these user interface aspects can fulfil the functional requirements, particularly the mandatory requirements, of a 3D modelling system. The system architecture overview will go deeper into how these functionalities, derived from the 2D analysis, are transitioned into the 3D system across all five identified subsets.

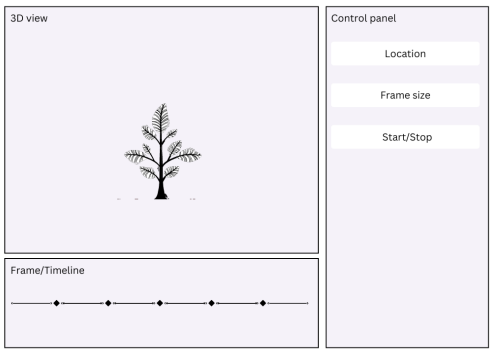

Figure 4.3: User interface draft

From the figure 4.3, it's evident that functionalities are organized into distinct sections on the design page. Scene-related adjustments are conveniently placed in the control panel for easy access, allowing users to modify and initiate the scene effortlessly. Two critical functionalities are predefined for the user: the plant's representation and the growth rules, which are both embedded with specific rules and conditions. These functionalities significantly influence the project's performance. Initially, simple shapes like cylinders represent the stems of plants to maintain simplicity; a plane indicates the garden's expanse and smaller planes represent leaves. For advanced prototyping, more detailed and realistic shapes will be considered based on the remaining project time. Lastly, the time of the system running is located at the bottom and utilized for setting conditional rules governing plant growth.

Chapter 5

Implementation and testing

5.1 Introduction

This chapter will focus on demonstrating the implementation of the project's design, presenting coding strategies, and the testing framework employed to validate the functionality and performance of the system.

5.2 Implementation

5.2.1 System setup

The project was implemented primarily within the Blender environment, utilizing Blender's built-in Python API for scripting and add-on development.

5.2.2 Hardware and Software environment

Hardware: The development was carried out on computers with adequate specifications to run Blender efficiently, including multicore processors and high-performance graphics cards to handle real-time 3D rendering.

Software: Blender version 3.4x was used, which supports extensive Python scripting and integration. This version was chosen for its stability and the rich set of features it offers for developers and artists alike.

5.2.3 Development Tools and Libraries

Blender Python API: Used for all scripting and automation within the project. This API provides access to Blender's functions, allowing for the manipulation of objects, scenes, and data within Blender.

External Python Libraries: Libraries such as NumPy and maths for numerical operations.

Version Control: Git was used for source code management, enabling version control and collaboration across the development team.

Database and Data Handling While Blender does not use a traditional database management system, data handling was a crucial part of the add-on.

5.2.4 Data Management

Internal Storage: Blender's own data-block system was utilized to store and manage data related to plants and garden layouts. Custom properties and groups were extensively used to manage the state and attributes of various entities within a garden simulation.

Data Integrity: To maintain data integrity, especially when updating or modifying garden layouts, transactional operations were simulated using Python scripts to ensure that all changes were either fully committed or rolled back in case of errors.

5.2.5 Integration

User Interface Integration: The add-on's user interface was integrated into Blender's existing UI. Custom panels and controls were developed to allow users to interact with the garden planning features directly within Blender's layout.

Interaction with Blender's Core: The add-on interacts directly with Blender's scene management, object manipulation, and rendering capabilities, ensuring that users can use the garden planning tools in conjunction with Blender's standard features like modeling, texturing, and rendering.

5.2.6 Implementation of Algorithms

One of the successful algorithms has been created to improve the functionality of the Blender add-on, with a focus on refining garden layout and plant growth simulation. This section will delve deeper into how these growth algorithms are implemented and how the interface is designed to facilitate user interaction.

Plant Growth Algorithm

A different algorithm from the L-system was implemented to simulate realistic plant growth based on genetic and environmental parameters. This algorithm uses procedural generation techniques to model plant development over time instead of replacement of existent objects in a scene.

Initially, the system was configured so that cylinder meshes, representing the primary parts of the stem, would expand every two frames. This growth interval is adjustable to accommodate the varying growth rates of different plant species.

```
def grow_mesh_handler(scene):
  if scene.grow_mesh_running:

    # other codes ....

    # grow main tree
    if scene.frame_current % 2 == 0:
        # codes growing main nodes

    if scene.frame_current % 4 == 0:
        # codes for grow branches
```

Listing 5.1: Growth rules in 'grow mesh handler' method

In Blender, the listing 5.1 is known as handler methods which is utilized within the Blender environment to manage various aspects of the virtual plant simulation. These methods were registered to respond to user inputs and system events, enabling dynamic updates and real-time adjustments to the simulation parameters. For instance, handler methods were registered to trigger growth animations, and modify growth protocols based on user interactions. The registration and unregistering of these handler methods facilitated the seamless integration of interactive functionalities within the simulation environment.The following figure illustrates how to facilitate the process when the handler is running:

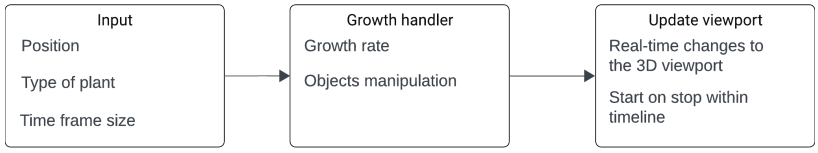

Figure 5.1: User interface draft

Input is located in the control panel as shown in the draft design of Figure 4.3 with other additional capabilities. Once the user selects the input options, the user can run the animation to trigger the grow mesh handler. This will update the simulation in real time and automatically execute the necessary actions and processes to create a simulation in the scene. However, when the user clicks the stop animation button, this handler will stop.

Rotation and meshes placement

The Growth handler and Input process start with the initial placement. This section explains the rotation process during the growth of certain plants' stems. Central to this process is the "rotate random" and shown in the figure 5.2 function, which utilizes Quaternion rotations to randomly influence the direction of the new stem's growth. Quaternions simplify the rotation tasks as they allow using the prior mesh as the rotation's axis. Within the code, the

coordinates (x, y, z) define the axis of rotation as shown in **??**, with the angle specifying how much rotation occurs around this axis.

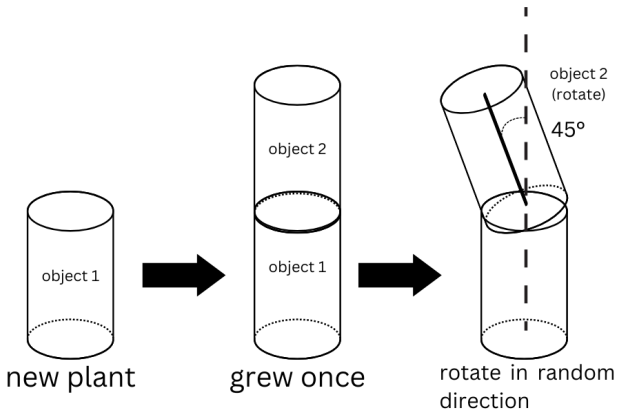

Figure 5.2: Process applying to rotate random

```
1  def rotate_quar(obj, new_obj, angle, axis):
2      quar_obj = obj.rotation_quaternion
3      direction = Quaternion(axis, angle)
4      rotation_matrix = direction.to_matrix()
5      new_obj.rotation_euler = rotation_matrix.to_euler()
6
7  def rotate_rand(obj, new_obj, angle):
8      min_value = -1.0
9      max_value = 1.0
10
11     x = random.uniform(min_value, max_value)
12     y = random.uniform(min_value, max_value)
13     z = random.uniform(min_value, max_value)
14
15     rotate_quar(obj, new_obj, angle, (x, y, z))
16     bpy.context.view_layer.update()
```

Listing 5.2: method rotate quartenion and rotate random

In Figure 5.2, object 2 is copied and move to top of object 1 and then applied the method rotate rand in listing **??** in line 7. The direction of rotation is randomly generate by the *random* libraries in Python, while rotate quaternion method shown in line 1 able to use object 1 as the axis of rotation for object two.

Rotation will be incorrect if object 1 is not used as an axis. This can be explain visually by looking at Figure 5.3:

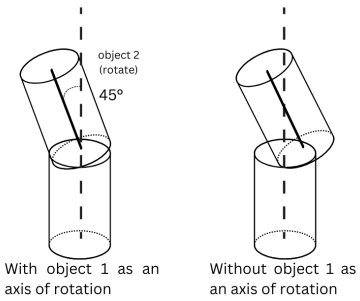

Figure 5.3: Axis rotation by quaternion

The 'random' module randomizes these axis coordinates, allowing for the plant to sprout in various directions.

Frame-Based Actions and Control

Planting a new plant is a simple process that mainly involves the user inputting the location and clicking a button to trigger the method outlined in listing 5.3:

```
def plant_new(x, y, location_z):
    """Create a new collection for the tree"""
    unique_suffix = str(uuid.uuid4())[:3]  # Take only the first 3 characters for brevity
    plant_type = bpy.context.scene.plant_type
    unique_name = f"{plant_type}_{unique_suffix}"
    new_collection = bpy.data.collections.new(unique_name)
    bpy.context.scene.collection.children.link(new_collection)

    radius = plants_rules[plant_type]["radius"]
    height = plants_rules[plant_type]["height"]

    add_cylinder((x, y, location_z), radius, height, new_collection)
    cursor_location_update((x, y, location_z - location_z))
    origin_to_cursor()
    all_objects = bpy.context.scene.objects

    # Deselect all objects
    for obj in all_objects:
        obj.select_set(False)

    print(f"Planted {new_collection.name} at ({x:.2f},{y:.2f})")
```

Listing 5.3: planting a plant

The listing 5.3 showcases a technique that has been implemented in one of Blender's panels. This method allows users to add a plant to their scene. The algorithm primarily focuses on ensuring that the plants are named distinctly and that their respective meshes are grouped into a single collection. Line 3 until 5 applies *uuid.uuid4()* to generate a unique identifier that looks like a long string of random characters. Turning it into the string using *str(uuid.uuid4())* converts this identifier into a string. Additionally, *[:3]* means "take the first 3 characters." This will provide a brief, unique suffix for naming the collection of objects that will be added to one tree with the name. A unique suffix can be used to help the system recognize different objects as the same plant. For example, if two new plants with the name "TREE" are added to the scene, the data handler must be able to differentiate between them. To achieve this, the two plants can be named "TREE 123" and "TREE cde", respectively. This way, the data handler can easily recognize which object belongs to which plant and uses a calculation of rotation in the previous section. Real-time adjustments between 9 through 15 will accurately add objects to prepare for growth.

After planting each plant, the simulation can be initiated from frame 1 and continued until the desired end frame is reached. Although the actual growth rate of plants may differ, in Blender, frames can be utilized as a timeline for measuring plant growth. Plant growth rate can be simulated by using conditional rules based on frames based on 5.1 in lines 7 until 11. For example, every 2 frames can represent 1cm of growth. If we use a ratio of 1 frame per second to estimate in real-time, we can assume that 2 frames passing means that 2 seconds have passed. For this project, we have decided to use 1 second to represent 1 week. To give an example, if we observe that the plant grows 1 cm every 2 frames, it means that the plant is growing 1 cm every 2 weeks. Currently, the system is set to default for 1 frame per second. If the frame rate is increased to 2 frames per second, the plant will grow 1cm every 2 frames. This means it will grow 1cm every week, since 2 frames equal 1 second.

In the growth handler, a frame control is also added to the method that adds a new node and a branch when a conditional rule is met. The conditional rule also applies the same frame detection as line 7 until 11 of 5.1.

```
1
2  def add_new_node(scene,collection):
3  # other codes: added new main stem to make initial stem higher
4
5  # below: conditional rule for branching
6  if scene.frame_current % 8 == 0:
7      print(scene.frame_current)
8      new_branch_collection = create_branch(scene,collection)
9      create_branch_object(new_obj, new_branch_collection)
10 else:
11     print("Don't add branch")
```

Listing 5.4: Growth rules in 'add$_n ew_n ode'$*method*

In listing 5.4 the same frame control is added specifically for branching purposes. Hence such stems could be developed such as below:

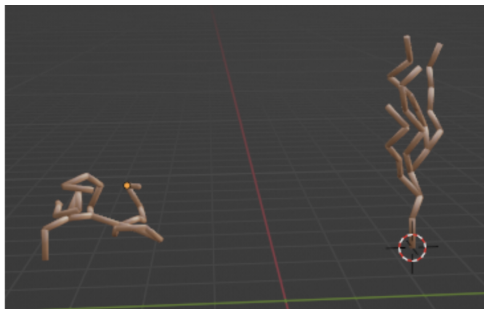

Figure 5.4: Growing stem branching every 2 frames, and growing every 1 frame. Left: 90 degrees, Right: 45 degrees

Specific plant type growth rules

In later development, the focus shifted towards tailoring growth rates based on the type of plant selected, with distinct frame-based rates developed using conditional rules. Unlike the L-system, which utilizes replacement rules, this project has been adapted to align with a timeline while incorporating random positioning and rotations of branches.

```
1
2  # Growth Handler
3  def grow_mesh_handler(scene):
4    if scene.grow_mesh_running:
5      collections = bpy.data.collections
6      plant_type = scene.plant_type
7      # Iterate over each collection
8      print(f"Current frame is: {scene.frame_current} ")
9
10     for c in collections:
11       # grow only main
12       if "TREE" in c.name and "Branch" not in c.name:
13         if scene.frame_current % 8 == 0:
14             print(f"Target plant: {c.name} ")
15             add_new_node(scene,bpy.data.collections.get(c.name))
16
17       elif "SHRUB" in c.name and "Branch" not in c.name:
18         if scene.frame_current % 4 == 0:
19             print(f"Target plant: {c.name} ")
20             add_new_node(scene,bpy.data.collections.get(c.name))
21
22     # same code structure for branches
```

Listing 5.5: Growth rules in 'grow$_m esh_h andler' method improved

The conditional rule in line 12 and 17 is adjusted to check if the collection name or objects are trees or shrubs for simplification in listing 5.5.

In later improvements, the system uses more actual plants to create scenes with distinct properties. The current version of the system can generate two different types of plants-sunflower, specifically known as 'Russian Giant' (Helianthus annuus 'Russian Giant') and baby's breath (Gypsophila paniculata) for prototype purposes. However, in the example provided, there is no actual annual growth or reflection of the properties of real plant growth. The system only provides an estimate based on the length of growth in certain weeks to show how different plants' growth is managed in this system.

```
1
2  plants_rules = {
3      "SUNFLOWER": {
4              "angle_of_branching": 0,  # degrees
5              "radius": 0.1,
6              "height": 0.5,
7          },
8      "BABYSBREATH": {
9              "angle_of_branching": 1,  # degrees
10             "radius": 0.01,
11             "height": 0.2,
12         },
13     # Add more plant types as needed
14 }
15
16 def get_plant_types(self, context):
17     # This could be dynamic based on some other data or a fixed list
18     types = [
19         ("SUNFLOWER", "Sunflower", "Yellow, tall, perennial flower, 12 weeks"),
20         ("BABYSBREATH", "Baby's breath", "Small plant and delicate flowers, 12 weeks"),
21     ]
22     return types
```

Listing 5.6: Dictionary and a description of labels for the plant's drop-down choices.

The current dictionary is not flexible enough to allow for easy addition of new plants based on a set of input parameters. This is because the growth rules are created for a single type of plant only, rather than being dynamically generated based on a set of rules that can be minimally adjusted by the system. For the 'SUNFLOWER' plant, the main branch will grow straight upward, while 'BABY'S BREATH' will branch out at a small angle with simple random branching that does not exactly reflect the real growth of the plant. The resulting image can be seen in Figure 5.5:

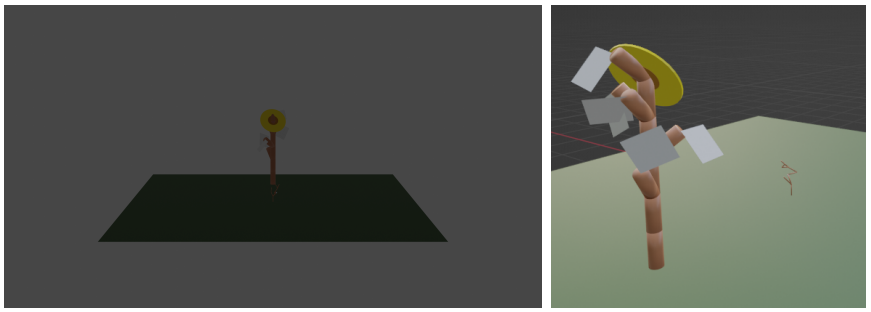

Figure 5.5: Sunflower bloomed at 12th frame and baby's breath, Left: Render view from camera looking at Z axis against X axis, Right: The viewport view

5.2.7 User interface

Control Panel Position

In the process of designing, it is essential to have easy access to the control panel for positioning and controlling the simulation. Figure 4.3 shows that the control panel is conveniently located on the right side of the viewport. In Blender, we can add such panels by defining a new panel class and inserting the space type as 'VIEW 3D,' which makes the panel available within 3D View. The implementation code is shown below:

```
1
2 class OBJECT_PT_PlantMeshPanel(bpy.types.Panel):
3     bl_label = "Garden Planning"
4     bl_idname = "OBJECT_PT_PlantMeshPanel"
5     bl_space_type = 'VIEW_3D'
6     bl_region_type = 'UI'
7     bl_category = 'Garden Planning'
8
9     # all button define below
```

Listing 5.7: Custom panel into 3D view

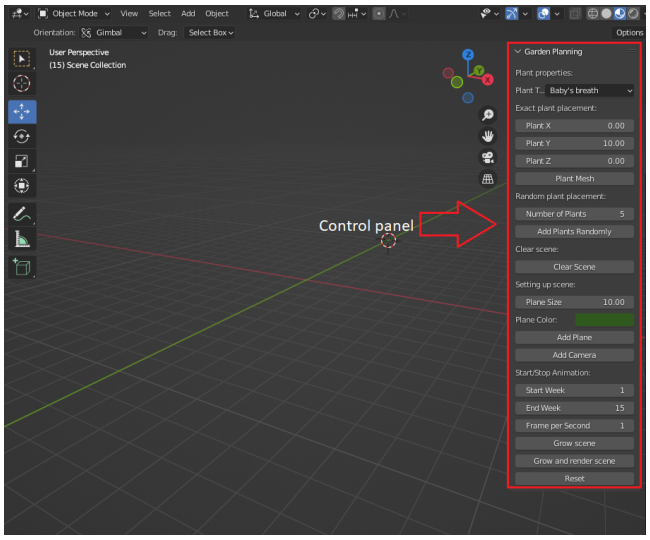

Figure 5.6: Panel located on the right side of Viewport

User Control and Pre-Action Warning System

Conditional rules are added to warn users when an action requires another object to exist first. Without a warning, the system cannot communicate properly and users may not know what's going on.

Two essential features of the system are the random placement of plants and the adjustment of the camera's position. These features depend heavily on the presence of a foundation plane, which serves as the basis for tree planting. To ensure the proper functioning of the system, a warning mechanism has been put in place to prevent users from taking any actions until a foundation plane has been established. The positioning of random plants is contingent upon the *Exact plant placement* input, which establishes the base location for where the random plants should be situated. Figure 5.7 shows how it will look like:

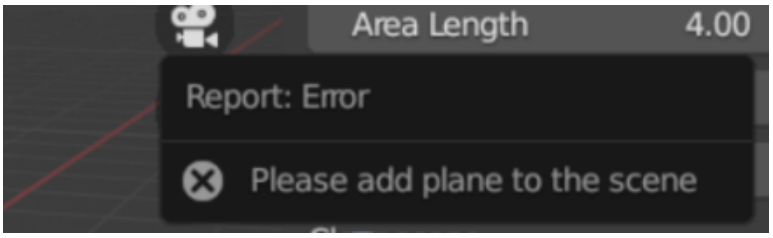

Figure 5.7: Warning when didn't add plane

```
1 if not plane_obj:
2      self.report({'ERROR'}, "Please add plane to the scene")
3      return {'CANCELLED'}
4
5 else:
6 # codes for adding plant at random location
```

Listing 5.8: Warning before adding camera or random plants

The camera setup also triggers a similar warning to ensure that calculations can be completed accurately. The camera is positioned at a distance, adjusted by a multiplier, and its height is incremented by a specified value. However, users cannot manually adjust camera positions but can adjust in the viewport if users are familiar with Blender's environment and control.

Users can select specific planting locations, modify the frame length, and adjust the plane size. For random planting, users can choose the number of plants, but this is limited to a maximum of five to maintain system performance. Users can also adjust the speed of rendering.

The limitations of the current software rise when only a limited amount of plant or animation can be done due to effect on performance.

Incorporating more than five plants may compromise the system's performance during growth simulations. While reducing the frame length might allow the simulation to complete, it may not be sufficient. Additionally, the speed of rendering and frame length have not yet been calibrated to accurately reflect the real growth rate of plants.

5.2.8 Code Organization

The following list outlines the key finalized files associated with this project:

1. **/images**: This directory contains all images used in the README file for documentation purposes.
2. **/texture**: This folder includes various texture files required for the graphical representation of plant models.
3. **final-version.blend**: The primary Blender file representing the final version of the main development environment.
4. **final-testing.blend**: A separate Blender file utilized for testing the add-on functionality and ensuring seamless integration with the 'garden-planning.py' script.
5. **garden-planning.py**: Version 1.0.1 of the add-on script containing specific rules for modeling the growth patterns of SUNFLOWER and BABYSBREATH plants. This script can be imported into any Blender environment.
6. **branching-prototype.py**: Version 1.0.0 of the prototype script focused on conditional branching rules to generate stem-like shapes. This file is also available as an add-on to any Blender environment.
7. **README.md**: Comprehensive documentation providing an introduction to the project's directory structure and guidance on usage.

5.2.9 Testing

Functional Testing

Table 5.1 shows the functional testing that has been done in this stage:

Table 5.1: Functional Testing

Test cases	**Expected result**	**Actual results**
Plant placement	Plants should be accurately positioned according to the specified type, scale, and location. If placement is set to random, the correct number of plants should be distributed within the defined area based on the specified width and height, ensuring random yet valid spacing.	Plants are placed accurately as per user settings for type, scale, and location. However, the placement lacks valid spacing when set to random, due to missing conditions for spacing validation.
Growth simulation	Each plant should grow according to predefined growth rules specific to its plant type. Growth should cease once the specified end frame is reached. Users must be able to set both initial and final frames for growth sequences. The growth rate of each plant should mimic the real-time growth characteristics typical of its species.	Plants are currently growing based on conditional statements linked to their type rather than a more flexible dictionary-based method, limiting customization and scalability of growth rules The simulation does not accurately reflect the real-time growth rates of plants, primarily due to constraints in understanding and implementing authentic growth parameters and time limitations.
UI responsiveness	Users should experience intuitive slider controls for adjusting values such as location, number of plants, plane size, frames per second, and frame range. A warning message should appear if an attempt is made to add a camera without an existing plane. Upon initiating rendering, the Image Editor should automatically open, displaying the rendering progress.	The user interface functions well with responsive sliders for adjusting plant location, quantity, plane dimensions, frames per second, and the animation frame range. The system correctly displays a warning when a camera is added without an existing plane. The Image Editor opens as expected when rendering begins, showing the progress of the render effectively.

The results will be discussed and analysed in detail.

Plant Placement As an easy example, we can use simple placement values in (x, y, z) coordinates, such as (0, 0, 0), (0, 1, 0), and (1, 0, 0), to demonstrate accuracy. This result is illustrated in Figure 5.8. Sunflowers were planted in three different positions and analysed from above. In this update, users are unable to change the scale of the object, as it has already been pre-defined to reflect the actual plant size and differentiate between different types of plants.

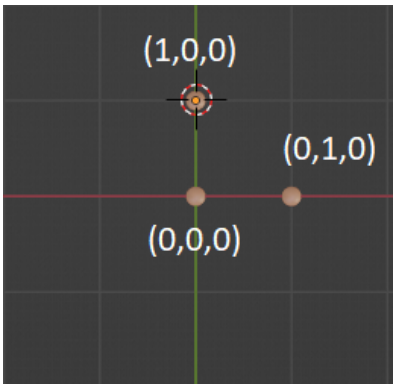

Figure 5.8: Point of view above, seeing three 'sunflower' objects in scene

Lack of spacing, when set random, does affect the logical growth of a real garden since plants should not intercept with each other. The result of the current system is shown below in Figure 5.9 and Figure 5.10.

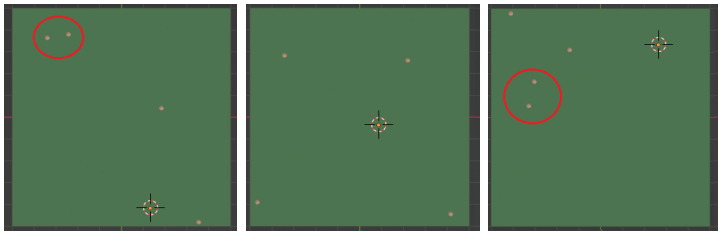

Figure 5.9: Random placement in plane of scale 10

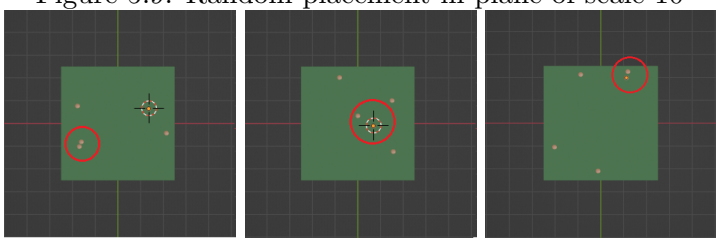

Figure 5.10: Random placement in plane of scale 5

Plants tend to grow closer together in smaller areas. This is due to a lack of Bresenham's line or plant interaction rules implementation, which can result in redundancy such as Figure 5.11.

Figure 5.11: Two plant models intersecting with each other. The grey plane represents leaves and the brown cylinder represents the stem

Growth simulation For testing purposes, we will use simple branching stems that do not follow any specific rules, except for branching at certain frames to see the current performance of the system especially the runtime of the code to complete the growth process. The testing will be done under the procedure below:

1. Preparation: Set up the Blender file, assign a plant number for each test, and prepare the stopwatch. Start the stopwatch at the beginning of the animation and stop it immediately at the end of the frame or animation. Repeat 3 times for each time, and calculate the average time taken in seconds for the animation to complete.

2. Methodology: Changes in FPS can affect simulated rendering growth if 1 sec equals 1 week. Below are the ratio changes. No rendering capabilities have been fully completed, this testing will not be performed as video, and we will fully rely on the frame conditional rule and viewport output. The details of the data collected to create the chart can be found in Appendix A.

   (a) Using FPS (1 second: 1 week):

      i. 1fps: If a plant grows 1cm every 2 frames, it means the plant is growing 1cm every 2 weeks.

      ii. 2fps: If a plant grows 1cm every 2 frames, it means the plant is growing 1cm every 1 week.

      iii. 4fps: If a plant grows 1cm every 2 frames, it means the plant is growing 1cm every 0.5 weeks.

3. Results:

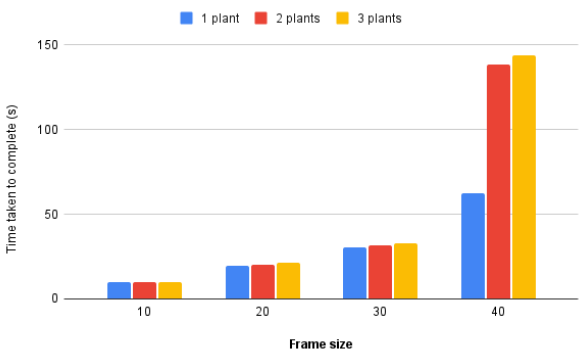

Figure 5.12: Impact of number of plants in the scene to the time taken for simulation to complete base

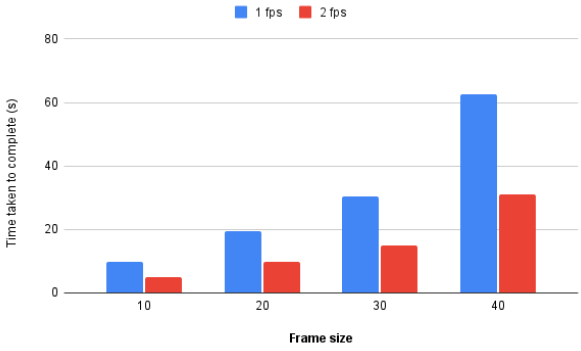

Figure 5.13: Effect of changing frame per second

The chart shown in Figure 5.12 reveals two important results. Firstly, adding more plants to the scene increases the runtime of the code required to complete the simulation for each frame. This is because every frame requires computation to move on to the next one, and adding more plants means more computation is needed, leading to potential delays. Secondly, the complexity of the system increases drastically when testing with 40 frames (equivalent to 40 weeks). To be considered feasible, the completion time should be almost the same as the time it takes to complete in seconds. Therefore, the system can only handle a maximum of 30 weeks of simulation if there is an ongoing growing plant with a complexity of growing every 2 frames and branching every 4 frames.

An analysis of two different frame rates is presented to demonstrate that, while maintaining the same level of complexity, it is possible to achieve a similar plant shape by growing two frames and branching every four frames, but with a faster completion time. Figure 5.13 reveals that higher frame rates result in significantly lower runtimes, which reduces the drastic change

experienced when using 40 frames. However, altering the frames per second will affect the plant's growth rate in the scene and should be adjusted accordingly.

Next, we will be conducting performance tests on two specific plants that have growth limitations: Sunflower and Baby's Breath represented in Figure 5.14. Currently, the dictionary only provides the height and radius parameters along with the name of the plants' preset growth rules. However, the plants are not accurately scaled and do not have perfect growth rates. The dictionary, as shown in listing 5.6, contains the radius, height, and angle of branching. It is assumed that sunflowers grow straight upwards and bloom at the 12th week, while Baby's Breath gradually branches out and adds flowers to random stems at the 12th week. Additionally, the growth of the plants can be extended beyond the current limit, but due to their programming, they are designed to grow only for a period of 12 weeks. This means that adding more frames will not have any effect on their growth.

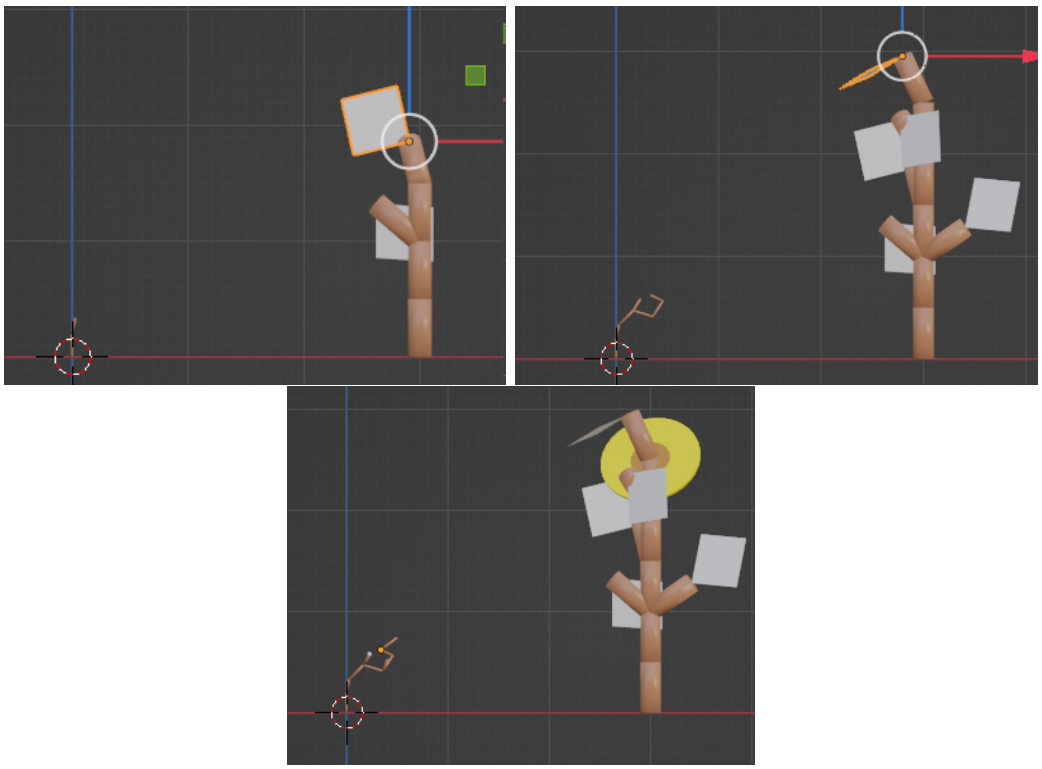

Figure 5.14: Baby's breath (left) smaller plant and Sunflower (right) larger plant growing at frame 5, 10, 12

Testing will involve adding plants to the scene and tracking completion until the 15th frame. A chart showing the simplification of values obtained is shown in Figure 5.15:

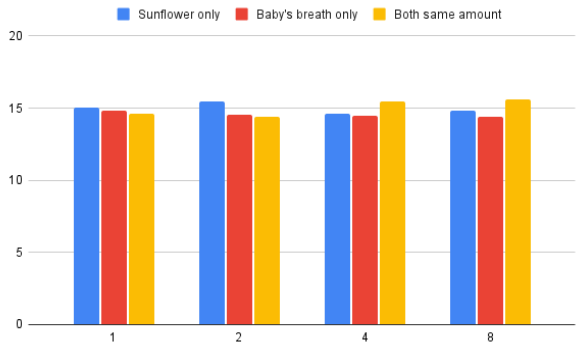

Figure 5.15: Different number of plants added gradually

In the figure shown in reference 5.15, it can be observed that there is not much difference in the time taken for completion. This may be due to the less complex rules for each plant. However, it seems that sunflowers take a bit longer to grow compared to a baby's breath. This could be due to the amount of mesh changes and location adjustments in the scene before moving to the next frame.

User-acceptance testing

User acceptance testing was executed internally through the simulation of common user scenarios to observe the system's functionality from an end-user perspective. During the tests, my role was limited to randomly planting two types of flowers using each prepared panel. Findings:

1. Ease of Use: The interface was found to be largely user-friendly, featuring intuitive controls and enough documented functions. Feedback from self-assessment highlighted the need for additional support for new users, particularly those unfamiliar with Blender. Consequently, recommendations include implementing a comprehensive tutorial or help section to facilitate a smoother onboarding process.

2. Functionality: The tool effectively fulfilled the primary design requirements. However, challenges were encountered with data persistence during session reloads, which have since been addressed and resolved. Continuous manual testing was conducted throughout the development process to refine functionality. Due to time constraints and the complexity of additional requirements, certain features have been deferred to future development phases.

3. Performance: Performance could be improved further. Nonetheless, it was noted that the add-on's responsiveness could be optimized when managing large-scale projects, such as extensive gardens with numerous plants.

5.2.10 Future resolutions

1. Randomization Control:

   - Challenge: The reliance on randomness for plant rotation and placement may lead to uniformity among different plants, reducing visual diversity in simulations, Figure 5.16.
   - Resolution: Specific growth rules for plants have been implemented in this project, but they need improvement to make them easier to follow.

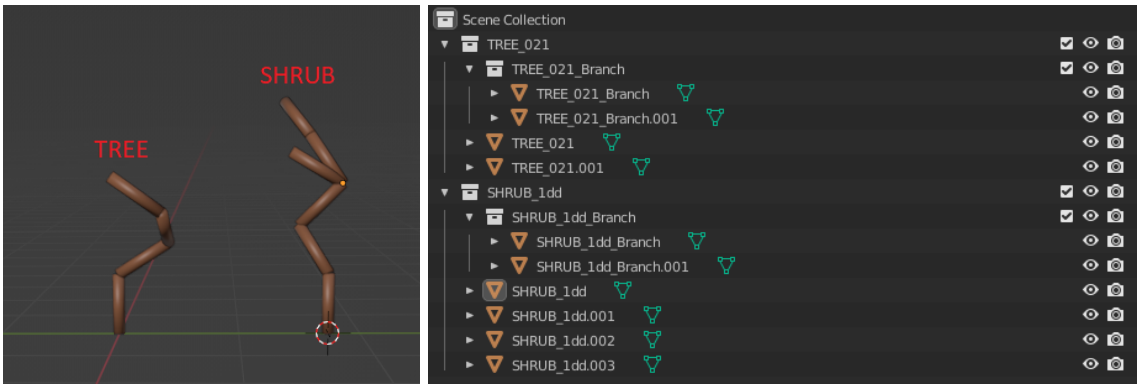

Figure 5.16: Two plants with the same size and angle, but different growth rates, highlight the need for diversity beyond just size and angle.

2. Performance Limitations:

   - Challenge: The system struggles with performance issues when more than five plants are added, potentially limiting the complexity of scenes that can be rendered efficiently.
   - Resolution: Optimize the underlying algorithms or upgrade hardware capabilities to handle more simultaneous processes without sacrificing performance. For random placement, temporarily limit the number of plants in the scene and avoid adding too much individually. Shown in listing 5.9 is an integer property to make sure the maximum plant to be added randomly is 5 and below.

```
    def register():
        global classes_registered
        if not classes_registered:
            # above: registration of other values....
            bpy.types.Scene.plant_count = bpy.props.IntProperty(
                name="Plant Count",
                description="Number of plants to add",
                default=5,
                min=1
            )
            classes_registered = True
```

Listing 5.9: Plant count registration

3. User Interaction with Camera Settings:

    - Challenge: The lack of manual adjustment options for camera settings restricts the user's ability to tailor views and angles, possibly impacting the usability and visual outcomes of the application.
    - Resolution: Develop a user interface that allows for camera position and angle adjustments, providing pre-set configurations for less experienced users and full customization options for advanced users.

4. Realism in Growth Simulation:

    - Challenge: The rendering speed and frame length settings do not currently reflect the real growth rate of plants, which can detract from the realism of simulations.
    - Resolution: Integrate growth models based on botanical research to more accurately simulate the real-time growth rates of different plant species.

5.2.11 Discussion

Feedback from the initial testing phase highlighted several areas for enhancement, particularly in refining the plant manipulation interface to offer a more intuitive user experience and expanding the biological accuracy of the plant growth simulations. However, substantial advancements in these areas were curtailed by constraints in time, skill, and available knowledge, limiting the scope of subsequent prototyping phases.

While plans for advanced prototyping included simulating seasonal changes and integrating complex meteorological data, these were not realized within the current scope of the project. The main challenges faced were:

1. Time Constraints: The project timeline did not allow for the extensive development required to implement and test advanced features comprehensively.

2. Skill and Knowledge Gaps: The complexity of integrating detailed meteorological data and simulating realistic biological growth patterns required a higher level of expertise in both software development and botanical sciences, which exceeded the current capabilities of the developer.

3. Resource Limitations: Limited access to advanced tools and libraries that could facilitate more sophisticated simulations and data integration also posed significant barriers.

To move beyond the initial prototyping phase and address the identified gaps, the following steps are recommended:

1. Skill Development and Collaboration: Engaging with experts in botany and software engineering to enhance the team's capabilities and refine the system's biological accuracy and technical robustness.

2. Securing Resources: Seeking additional funding or partnerships to acquire advanced technological tools and resources necessary for sophisticated simulations.

3. Enhancements to Existing Solutions: The alternative approaches selected could be further refined and improved with the availability of more advanced software solutions.

The justification for not proceeding beyond initial prototyping lies primarily in the need to ensure a solid and reliable foundation before introducing more complex features. Ensuring the basic system is robust and user-friendly paves the way for future enhancements without compromising the system's core functionality and user experience.

Chapter 6

Results and discussion

6.1 Goals achieved

- **1.1** The system supports plant growth with constraints on the number of frames and time. While it doesn't strictly use an L-system for growth, it employs a similar concept utilizing dictionaries to manage various growth parameters and distinguish different plant types.
- **1.2** The system is currently able to demonstrate how data might be organized within dictionaries, yet widespread implementation is lacking. Differences are noted only in growth rates, while achieving distinct sizes, shapes, and other botanical features like leaves, flowers, and fruits remains unaccomplished due to the complexity of biological information required.
- **1.3** Despite examining the literature on Bresenham's algorithm, its application within Blender is not well understood, leading to issues with plants intercepting each other.
- **1.4** By using frames and setting the frame rate to 1 fps, the simulation portrays each second as equivalent to one week. While this method fulfills the project's current technical requirements, further improvements could enhance realism.
- **2.1** Users can select various aspects of plant life such as location and type, and even add random placements. However, accurately representing plant maturity or age is not yet feasible, as initial plant shapes needed to simulate advanced age are not generated behind the scenes.
- **2.2** This goal is achievable under the condition that users first define the area, and the number of plants is limited to five, due to performance constraints leading to extended processing times.
- **2.3** Users have the flexibility to navigate through the simulation in Blender and render animations of the scene, although these actions cannot be performed simultaneously.

No.		Requirements	Importance	Status
1. Plant's growth	1.1	System can grow individual plant with L-system rule	Mandatory	C
	1.2	System can grow plants according to specific plants' rules in the library of growth rules of each plant	Mandatory	NC
	1.3	Plant should refrain from branching when the surrounding space occupied, demonstrating plant-to-plant intersecting	Desired	NC
	1.4	Plant growth should be visualised weekly	Mandatory	C
2. User interaction	2.1	User can pick plant's type, age, and location	Mandatory	C
	2.2	User can place multiple plants in a scene	Desired	C
	2.3	User can interact with the scene and plants in real-time during the simulation	Desired	C
	2.4	Users can define the absolute run time or pause	Desired	C
	2.5	User can change parameters of seasons such as temperature, the intensity of light, seasons and weather	Optional	NC
3. Seasonal effect	3.1	The system can simulate all four seasons during runtime. The seasons are easily distinguishable from each other.	Desired	NC
	3.2	The system accurately visualizes changes in the scene's background to seasonal variations, temperature, and light intensity parameters.	Optional	NC

Table 6.1: Project requirements, C - Completed, NC - Not Completed

- **2.4** Two buttons are provided, allowing users to start or stop the simulation or to run and render simultaneously. Each button offers control over the simulation's operation.
- **2.5** As the system does not yet accommodate seasonal changes, no progress in this area can be reported.
- **3.1** and **3.2** No substantial progress has been made, although there is a preliminary concept of adding texture animations, such as depicting falling flowers or dots under a plant to represent flower petals.

6.2 Further works

The number of completed and unfinished requirements is almost the same. Therefore, there is still a significant amount of work to do within the given timeframe. However, there is room for improvement in the future to complete the Garden Planning System as a whole.

Some important works that should be highlighted are:

- **Extend Dictionary Use:** Enhance the use of dictionaries to include multiple attributes per plant, such as growth rate, lifespan, branching angles, and other key biological properties that define various plant species.
- **Performance Enhancement through Algorithms:** The current limitation on the number of plants hampers the system's utility for real-world garden planning. It is advisable to explore different algorithms and perhaps more robust computational environments to boost system performance.
- **Increased User Control:** Provide users with greater flexibility by allowing adjustments in plant sizes, management of planting areas, pruning activities, and other gardening tasks to enhance the interactive experience.
- **Enhance Realism in Plant Modeling:** To improve the visual realism of the simulation, incorporate detailed models for leaves and flowers instead of using simplistic geometric shapes like cylinders. This would provide a more lifelike and visually appealing representation of plant growth.

Chapter 7

Conclusions

In summary, this research enhances our knowledge of simulating plant growth through sophisticated 3D modeling tools such as Blender. It demonstrates that with the use of algorithms and frame control, simulated plant growth can effectively replicate real-world conditions, proving valuable in professional landscape design applications. Future research could investigate incorporating environmental impact factors and the scalability of simulations to encompass larger ecosystems. Although some goals of this study were not met due to various challenges encountered during the research, the experience has been enriching and provides insights into potential expansions of this project.

Bibliography

[1] Bresenham's Line Generation Algorithm - GeeksforGeeks — geeksforgeeks.org. https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/. [Accessed 03-12-2023].

[2] Will a Tree's Branches Rise as It Grows? — Trav's Tree Services — travstrees.com.au. https://www.travstrees.com.au/information-centre/will-a-trees-branches-rise-as-it-grows. [Accessed 18-11-2023].

[3] UK flooding: Heavy rain causes flooding in parts of England and Scotland, 10 2021.

[4] Alslandscaping. 5 Reasons Why 3D Garden Design is a Great Idea!, 2 2023.

[5] Bilas, R. D., Bretman, A., and Bennett, T. Friends, neighbours and enemies: an overview of the communal and social biology of plants. *Plant, Cell & Environment 44*, 4 (2021), 997–1013.

[6] Boudon, F., Pradal, C., Cokelaer, T., Prusinkiewicz, P., and Godin, C. L-py: An l-system simulation framework for modeling plant architecture development based on a dynamic language. *Frontiers in Plant Science 3* (2012).

[7] Bresenham, J. Algorithm for computer control of a digital plotter. *Seminal graphics: pioneering efforts that shaped the field* (1965).

[8] Dejong. Theodore M., Da Silva. David, V. J. E.-G. A. J. Using functional structural plant models to study, understand and integrate plant development and ecophysiology. *Annals of Botany 108* (October 2011), 987–989.

[9] Eleanor Webster, Ross Cameron, A. C. Gardening in a changing climate.

[10] Hemmerling, R., Kniemeyer, O., Lanwert, D., Kurth, W., and Buck-Sorlin, G. The rule-based language xl and the modelling environment groimp illustrated with simulated tree competition. *Functional Plant Biology 35*, 9-10 (2008), 739–750.

[11] Hubert Krawczyk, K. B. Imperfect plants, 2021.

[12] J. Vos, J.B. Evers, G. H. B.-S. B. A. M. C. P. H. B. d. V. Functional–structural plant modelling: a new versatile tool in crop science. *Journal of Experimental Botany 61* (May 2010), 2101–2115.

[13] Leopold, N. Algorithmic botany via lindenmayer systems in blender, Aug. 2017.

[14] Lindemayer, A. Mathematical models for cellular interactions in development ii. simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology 18* (March 1968), 300–315.

[15] Magazine, B. G. W. Gardening for beginners: Plant types explained, 9 2022.

[16] Prusinkiewicz, P., and Lindenmayer, A. *The Algorithmic Beauty of Plants*. Springer Science Business Media, 1990.

[17] Prusinkiewiczy, P., Hammely, M., Hananz, J., and Mech, R. L-systems: From the theory to visual models of plants. *2nd CSIRO Symposium on Computational Challanges in Life Sciences* (1996), 28–29.

[18] Room, P., Maillette, L., and Hanan, J. Module and metamer dynamics and virtual plants. *Advances in Ecological Research 25* (1994), 105–157.

[19] Stevenson, A. Gardens: A garden for all seasons.

[20] Wolfie, M. Planning a four-season garden? here's what to consider, 2021. Last accessed 3 October 2021.

[21] Yunchen Wang, K. B. The imperfect plants, 2019.

Appendices

Appendix A

Appendix A: Tests data

Fps	N. plant	Max. frame	Run 1 (s)	Run 2 (s)	Run 3 (s)	Average runtime (s)
1	1	10	9.75	9.60	9.69	9.68
		20	19.50	19.40	19.45	19.45
		30	30.49	30.16	30.31	30.32
		40	62.58	62.96	61.80	62.45
	2	10	9.68	9.47	9.73	9.62
		20	20.58	20.48	20.40	20.45
		30	31.77	31.47	31.44	31.56
		40	138.96	138.36	138.28	138.45
2	1	10	5.37	4.75	4.61	4.91
		20	9.92	9.82	9.90	9.80
		30	14.68	14.84	14.68	14.82
		40	31.11	31.03	31.08	31.07

Table A.1: Comparing Program Runtime with Different Frame Sizes, Number of Plants and FPS

Test of specific plants:

Number of plants	Runtime (s)
1	15.05
2	15.48
4	14.65
8	14.82

Table A.2: Only Sunflowers

Number of plants	Runtime (s)
1	14.80
2	14.55
4	14.46
8	14.42

Table A.3: Only Baby's breath

Number of plants	Runtime (s)
1	14.59
2	14.38
4	15.46
8	15.60

Table A.4: Same number of both

Three samples of a single plant showing randomness of output plant when were tested at different frames at 1fps.

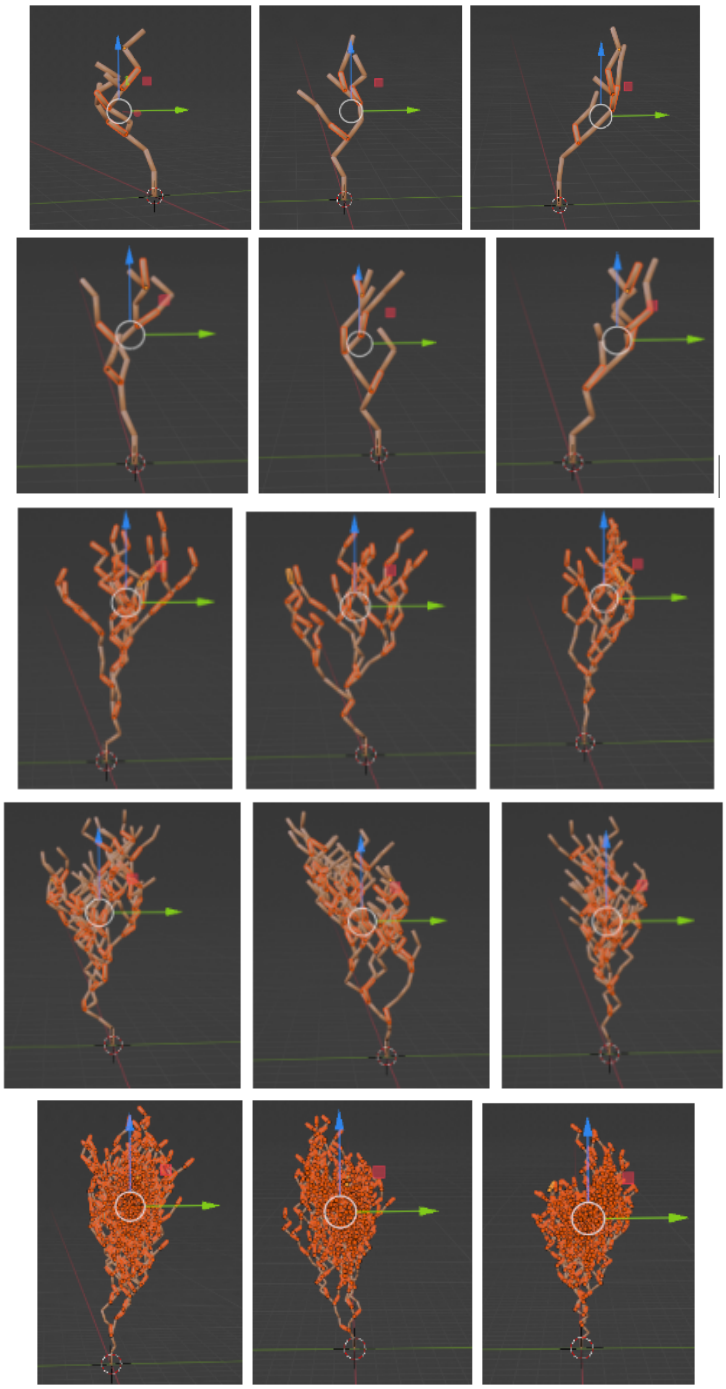

Figure A.1: Sample images of testing