EG UK Computer Graphics & Visual Computing (2024)
A. Slingsby and D. Hunter (Editors)

Real-time Data-Oriented Virtual Forestry Simulation for Games

B. Williams[1], T. Oliver[1], D. Ward[1] and C. Headleand[1]

[1]Staffordshire University Games Institute
School of Digital, Technologies, Innovation and Business
Staffordshire University, UK

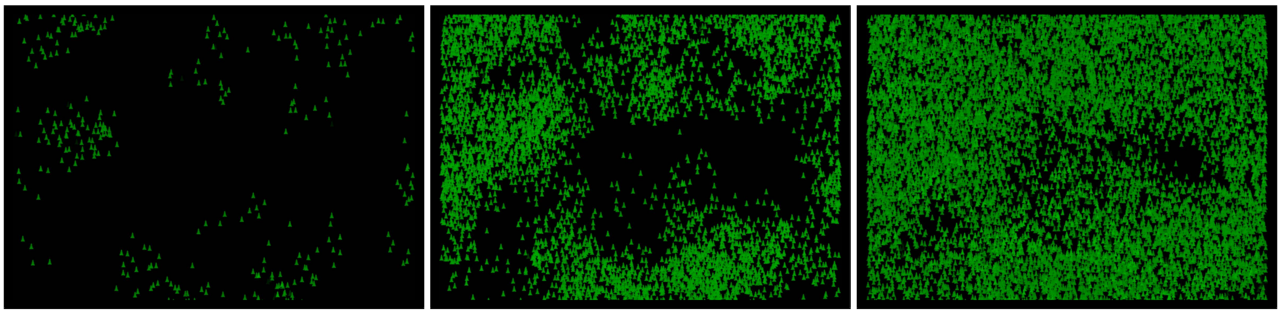


Figure 1: *The growth and development of a virtual forest, simulated in real-time, utilising our optimisation strategy.*

Abstract

The current frontier of virtual forestry algorithms remain largely unoptimised and ultimately unsuitable for real-time applications. Providing an optimisation strategy for the real-time simulation of virtual forestry would find particular utility in some areas, for example, in video games. With this motivation in mind, this paper presents a novel optimisation strategy for asymmetric plant competition models. In our approach, we utilise a data-oriented methodology with spatial hashing to enable the real-time simulation of virtual forests. Our approach also provides a significant improvement in performance when contrasted with existing serial implementations. Furthermore, we find that the introduction of our optimisation strategy can be used to simulate hundreds of thousands of virtual trees, in real-time, on a typical desktop machine.

CCS Concepts
• ***Computing methodologies*** $\rightarrow$ ***Artificial life; Parallel algorithms;*** *Computer graphics;* • ***Software and its engineering*** $\rightarrow$ *Interactive games;*

1. Introduction

Procedural Content Generation (PCG) has been applied in recent years to automatically generate a wide range of content. There are several benefits to the automatic generation of content in video games. For example, a primary motivator for its usage is the significant reduction in development time in contrast to manual design approaches. The benefit is emphasised when modelling systems with thousands of individual components, such as virtual forestry, which ordinarily would take a substantial amount of time to create. The possibility of generating complex systems inspired the earliest uses of PCG in games, circumventing the hardware limitations of early computer systems. One of the first adopters of PCG was the space-based trading game *Elite*, which automatically generated in-game solar systems with thousands of stars [HMVDVI13]. This motivation has inspired several authors to explore and develop solutions focused on the generation of virtual forestry in games. The majority of approaches focus on stochastic point sampling, which

DOI: 10.2312/cgvc.20241218

in some cases has shown to be successful in generating believable forestry [WRH19]. These distribution methods offer a computationally efficient solution for forestry generation, at the expense of model accuracy. This is contrary to plant competition models, which simulate the individual growth, spread, and death of trees in a plant community. Such algorithms model factors such as equidistant spacing of trees as emergent properties of the simulation itself. For example, Field of Neighbourhood (FON) based approaches [BWB*04] embody equidistant spacing by enforcing spatial competition rules. These types of bio-inspired models are computationally expensive due to the interdependence of forest growth, but provide a high-fidelity model of forest development.

A limiting factor of existing plant competition models and their adoption concerns their computational cost. For example, video games are inherently real-time experiences and demand minimal computational overhead in algorithm design. Whilst several algorithms exist for simulating forest growth, the majority are not explored in the context of games. Instead, most papers surrounding this topic consider forest growth as an offline model, which has use cases in ecology-based research. However, considering these growth models in real-time simulation has largely been overlooked. Current plant competition models are largely unsuitable for in-game usage, due to their computational expense and frame latency. As such, the urgency for a real-time, efficient, and optimised forest simulation approach is stressed. It is with this pressing need in mind that this paper explores the creation of a highly efficient optimisation strategy for FON-based forest growth models in the popular Unity 3D games engine. In our strategy, we leverage the power of the relatively new Entities package, utilising the Entity-Component System (ECS) for the parallel simulation and generation of virtual forestry. Optimisation is largely performed through the adoption of a data-oriented ECS, the application of spatial hashing, the vectorisation of simulated entities, and the mass parallelisation of the agent-based simulation using the Burst compiler and Job system. We also explore the consideration of real-time and continuous forest growth, which remains an unexplored area in video games.

It is worth noting that in our previous work we have considered the procedural generation of forestry in detail. In our initial work we introduced and evaluated simple point distribution methods against plant competition models in terms of believability [WRH19]. Amongst several interesting findings, we found that bio-inspired plant competition models generated forestry ranked as generally more believable [WRH20]. In this paper, we intend to focus on the optimisation of these plant competition models for real-time generation.

2. Background

The body of PCG-related literature has grown to consider a wide range of applications from its inception, such as the automatic synthesis of textures [DLY*20], real-time generation of entire game levels [KCR*23] or the generation of virtual settlements [WH17]. One aspect which has been considered in this area concerns the generation of virtual vegetation, such as trees or shrubs. In early work, Lindenmayer et al. pioneered the use of L-Systems to model the growth and dichotomous branching structure of algae plant systems [PH13]. L-Systems were quickly adapted by Aono and Kunii to model the monopodial branching structure of trees, producing the earliest generated tree images with an L-System [Pru86, AK84]. Since their inception, L-Systems remain a common approach for the generation of virtual tree structures. In recent research for example, machine learning (ML) algorithms are trained with generated L-Systems and subsequently used for virtual tree generation [LLB23]. Another example considers using L-Systems to distribute virtual fruit on generated trees, a previously unexplored topic [DeJ22]. However, most generative L-System approaches in this area continue to target the generation of virtual trees and their skeletal structure [TTWZ20, Ste23, Ber21]. Some papers have focused on other aspects of tree generation, such as the procedural generation of bark-like textures [VRP22] or the generation of complex root structures [LKL22].

Outside the scope of generating individual plants, another topic concerns the generation of plant communities. This problem was initially tackled by Reeves and Blau [RB85], who introduced a novel use of particle systems to model individual trees in a virtual forest. The approach utilises designer-specified parameters to distribute trees in a virtual environment. Point distribution methods, outside of Reeves and Blau's approach, also prove to be a popular category of algorithm in the literature. Several papers have shown their utility in procedurally distributing objects in scenes, including the distribution of trees and forestry [LD05]. Ecormier et al. [ENMGC19] show a recent example of this, using a variance-aware disk-based distribution model to generate virtual forest scenes.

Other algorithms which model ecological development through plant competition have been proposed in previous research [FS18]. Plant competition models consider the individual simulation of plants in an ecosystem. Within these models, the competition between species is a core focus. One of the initial methods which adopted this approach was put forward by Bauer et al. [BWB*04], in which a field-of-neighbourhood (FON) model is proposed. The FON is a radius situated around each plant, determining the zone in which this tree competes with others in the system. If for example, two tree's FON radii overlap, they are considered in competition with each other for resources. An example illustrating this paradigm can be seen in Figure 2.

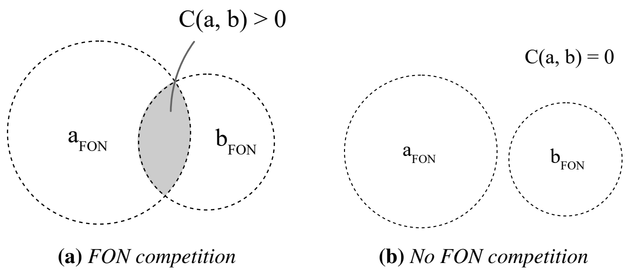


Figure 2: *A diagram showing FON-based asymmetric competition. On the left, plants a and b's FON radii overlap, and they are in competition for resources. Conversely on the right, a and b's FON radii do not overlap, and there is therefore no competition between the two.*

Alsweis and Deussen classify plant competition into two categories: asymmetric and symmetric competition [AD15]. Symmetric competition considers that resources are split evenly between two plants in competition. In contrast, asymmetric competition, resources are split unevenly between the two. In asymmetric competition, the virtual tree with a larger FON will dominate competitors with a smaller FON [AD15]. This models canopy coverage occluding sunlight to younger plant species who are in the vicinity of each other. Alsweis and Deussen use bio-inspired rules coupled with the FON model to generate plant systems, using an asymmetric competition model. Lane and Prusinkiewicz [LP*02] use a similar approach to generate realistic plant communities. The authors use a multi-set L-System in the generation of trees in the system. Additionally, an approach similar to Alsweis and Deussen's [AD15] asymmetric FON model is used, simulating individual trees and determining competition using a radius-based model.

Simulating forestry via bio-inspired plant competition models also proves to be a well-researched topic in ecological literature. The development of simulation software such as GreenLab [HDRZ*03] enables ecologists to study and predict the development of plant communities. In recent work, de Reffye et al. [DRHK*21] summarises the use of GreenLab over two decades since its inception, showcasing its use as an effective tool for ecological research. Outside of predicting plant community growth, some authors have previously utilised GreenLab for procedurally generating forest scenes. For example, Cournede et al. [CGB*09] use GreenLab in a novel method of distributing trees in a virtual forest scene. Another area concerns the optimisation of real-time generation and simulation of virtual forest scenes. The motivation for an efficient method of generating forestry has been stressed by previous authors, especially in the context of real-time applications [KGM14]. The optimisation of rendering forestry has been considered previously in the literature, especially in the context of games. For example, the SpeedTree SDK is one of the front-running products used in video games for the real-time generation and rendering of trees [FJFJ17]. The SpeedTree SDK also enables the management of millions of individual trees through its *Forest* library, enabling real-time usage in games [spe]. However, the optimisation steps focus solely on the rendering process; efficient generation and real-time growth of virtual forestry is not considered by the library. This is also seen in the literature, with the majority of optimisation approaches focusing on the rendering of large numbers of tree instances [BN12, RB85, BLZ*11]. One technique, for example, uses different Level of Detail (LoD) meshes which are swapped in real-time depending on camera distance, to dramatically reduce render time [BLZ*11].

Some researchers have considered the problem of optimising forest growth and simulation in the literature. For example, Kohek and Strnad [KS15] leveraged the GPU to parallelise the synthesis of 3D tree models and their distribution within an environment. The approach was capable of generating realistic forests with more than 500 trees in a second. The approach however, consumed a large amount of memory, even for smaller forest sizes. Similarly, Lipp et al. [LWW10] considered the massive parallelisation of generating L-Systems for virtual tree branching. Kohek and Strnad [KS18] propose a novel method which utilises particle flow simulation to generate and render forest scenes. The authors, however, do not consider the problem of optimising plant competition models. A similar optimisation strategy was later considered by Carey [Car19], leveraging L-System instancing to optimise the generation process.

In recent work, Newlands and Zauner [NZ22] present an open-source tool for generating forest scenes, focusing on the optimisation of the rendering process. The approach utilises plant competition models, but does not focus on the optimisation of the algorithm for real-time simulation in games. Closely following on from the work of Zauner, Badr et al. [BHRdA24] utilise a data-driven approach to massively optimise the visualisation task of real-world forestry. However, the authors only consider the optimisation of the rendering process. On a similar note, Kenwood et al. [KGM14] introduce an efficient and highly optimised approach for generating forestry scenes, but do not consider simulating plant competition and forest growth. Nunes et al. [NFP22] showcase a novel method of synthesising rich forest scenes to train an ML-based algorithm. The authors consider some optimisation steps, but ultimately overlook the simulation of trees in a plant competition model. Garifullin et al. [GFK21] present an optimisation method for modelling large-scale plant communities, such as forestry. The optimisation step uses approximate instancing, in which plants are grouped together and instanced throughout the scene. Despite the prevalence of plant competition simulations in the literature, there are currently no real-time optimisation strategies focusing solely on the simulation of plants. The majority of optimisation strategies either focus on single-pass generation [KS18, Car19] or optimising the rendering process [KGM14, BHRdA24]. Providing optimisations for simulating plant communities would enable several benefits to researchers and game developers alike. For example, real-time simulation would offer the possibility of concurrent forest growth in games, enabling more adaptive and immersive environments. Providing real-time optimisations would also make the study of plant competition models easier, fostering research in this area.

3. Optimisation Strategy

Our optimisation strategy primarily focuses on the application of a data-oriented Entity Component System (ECS) to massively parallelise FON-based plant competition algorithms. The Unity games engine was utilised for two reasons. Firstly, Unity is a commonly used and front-running games engine both in the literature and the commercial sector. Secondly, Unity offers an attractive built-in data-oriented technology stack (DOTS) for optimising large-scale simulations such as simulating virtual forestry. To further reduce simulation latency, we leveraged the *Burst* compiler to translate C# Intermediate Language (IL) bytecode to highly optimised native instructions. The Unity C# job system was also utilised to concurrently execute parts of the simulation and perform large-scale optimisation. A diagram highlighting the usage of the job system within the ECS architecture is shown in Figure 3.

In our optimisation strategy, the plant competition simulation is split into three primary tasks, following the Lane and Prusinkiewicz model [LP*02]:

1. **Succession**: Simulated trees grow and age each tick of the simulation. Once trees reach a certain age they die, and are subsequently removed from the simulation.
2. **Plant propagation**: Trees reproduce similar to the method proposed by Alweis and Deussen [AD15]. Child trees are spread

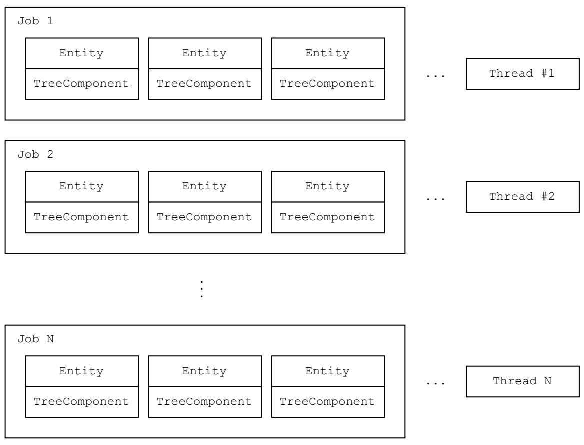


Figure 3: *A diagram showcasing the utility of the Unity C# job system, enabling the easy batching of entity processing across several threads. In the ECS, entities are stored contiguously in memory, enabling easy concurrent vectorisation for parallel processing.*

locally around the parent tree, helping to cluster trees together of the same species. Trees can only spread their seed when they reach a mature age.

3. **Self-thinning**: This step embodies the competition between plants. If the FON of one plant is within distance of another, the two plants are considered in competition. The plant with a larger FON ultimately dominates the smaller plant, which is removed from the simulation.

These steps embody the bio-inspired plant competition model discussed in our previous work [WRH19]. Trees are individually simulated and plant competition is assessed using Alsweis and Deussen's [AD15] asymmetric competition rule:

$$I(a,b) = \begin{cases} C(a,b) & \text{if } a_{\text{FON}} > b_{\text{FON}} \\ C(a,b) \text{ or } 0 & \text{if } a_{\text{FON}} = b_{\text{FON}} \\ 0 & \text{if } a_{\text{FON}} < b_{\text{FON}} \end{cases} \tag{1}$$

where $C(a,b)$ gives the competition/FON-overlap between the two plants a and b. This competition rule embodies an uneven split of resources between the two plants, based on which FON radius is larger. The plant with the smaller FON radius will be dominated by its competitor, reducing its access to resources and its eventual death, upon which it is removed from the simulation. In the simulation, a wind direction and magnitude are also present, which are randomised each tick, as in our previous work [WRH19]. This variable is present in the simulation to more accurately model the environmental forces involved in seed distribution. The asymmetric plant competition model used in our simulation closely follows Alsweis and Deussen's approach [AD15]. Competition is represented using FON radii, which is directly correlated to a tree's age – larger trees have larger FON radii, and dominate smaller trees for resources. Other environmental variables, such as soil acidity or air quality are not taken into consideration. Instead, our algorithm provides a simple model of plant growth in a virtual environment. We plan to investigate more comprehensive models which consider biological factors in future work.

3.1. Entity-Component System Architecture

The *Entities* package was used with Unity 2022.3.4f1 to provide access to the in-engine data-oriented technology stack and ECS. The overarching paradigm of ECS is heavily focused on data-orientation, organising the simulation around the memory footprint of simulated entities and their archetypes. Entities can have attached components, which represent data associated with entities. Unlike the traditional object-oriented paradigm, entities are simulated holistically by a system, which govern the simulation for a category of an archetype of entities. In our ECS architecture, virtual tree entities were tightly packed in contiguous memory to a) avoid internal memory fragmentation and b) enable easy vectorisation for concurrent parallelisation with the job scheduling system.

A single ECS system provides the simulation for the virtual forest, which acts as a container for all simulated tree entities. Two unmanaged ECS components, `ForestComponent` and `TreeComponent` were used to represent simulated trees and forests in the simulation. Entities with an attached `TreeComponent` instance are selected by the overarching system and updated every tick of the simulation. Component tags were avoided in classifying mature or dead trees, to avoid structural changes and improve overall efficiency. Furthermore, entities with attached `TreeComponent` instances are associated to overarching `ForestComponent` instances via an unsigned integer index. A diagram outlining the main system used for forest simulation can be seen in Figure 4.

To provide concurrent structural changes, i.e. adding or removing virtual trees from the simulation, an Entity Command Buffer (ECB) data structure was used. The use of an ECB enables thread-safe structural changes to be recorded across parallel jobs and played back at a safe state. This enables jobs to run concurrently across several threads, eliminating the need for serial execution and improving simulation performance.

3.2. Jobs

As seen in Figure 4, a number of separate jobs are scheduled to perform the steps of the plant competition algorithm. These jobs are scheduled in parallel using the Unity C# job system to multithread the solution. Whilst jobs are computationally distributed across threads, jobs are sequenced in serial. For example, Job #1 is distributed in parallel but will always finish before Job #2. This enables the safe synchronisation of entities within the ECS at a given point in time. Jobs are parallelised by inheriting from the `IJobEntity` class, enabling jobs to be automatically distributed across entities. The first job to run each tick, `AssignSpatialIndexJob`, iterates over all tree entities in parallel and assigns them a hash value based on their position. More details of this process are discussed in the following section. The next scheduled job, `CullDeadTreesJob`, simply iterates over each tree entity currently flagged as dead. It then removes them from the simulation. This process is separated into a scheduled job as it en-

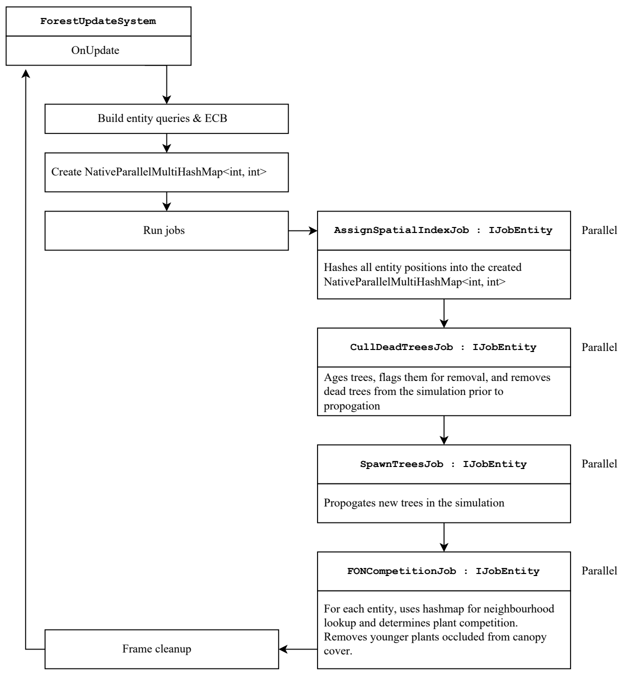

ForestUpdateSystem
OnUpdate
Build entity queries & ECB
Create NativeParallelMultiHashMap<int, int>
Run jobs
AssignSpatialIndexJob : IJobEntity
Parallel
Hashes all entity positions into the created NativeParallelMultiHashMap<int, int>
CullDeadTreesJob : IJobEntity
Parallel
Ages trees, flags them for removal, and removes dead trees from the simulation prior to propogation
SpawnTreesJob : IJobEntity
Parallel
Propogates new trees in the simulation
FONCompetitionJob : IJobEntity
Parallel
For each entity, uses hashmap for neighbourhood lookup and determines plant competition. Removes younger plants occluded from canopy cover.
Frame cleanup

Figure 4: *An overview of the `ForestUpdateSystem`, which performs the virtual forest simulation. Each job is scheduled in sequence but distributed in parallel, leveraging multi-threading to significantly optimise the batch processing of virtual trees.*

ables the safe replaying of large structural changes to the ECS at a well-defined point. This job is followed by the scheduling of the `SpawnTreesJob`, which selects all mature plants and propogates new plants from their position. For each mature tree, a normalised probability is randomly selected as $p \in [0,1]$. If $p < t$, where t is simulation-wide propogation probability, then the tree sows its seed and a new plant is added local to the tree's position. A new propogated tree position $\mathbf{p}'$ is selected with $\mathbf{p}' = \mathbf{p} + \mathbf{w}d_s$ – where $\mathbf{p}$ is the parent tree position, $\mathbf{w}$ is the current wind vector and d_s is the randomised forest-wide spread distance for new plants. The final job, `FONCompetitionJob`, iterates over trees and considers others local to itself for competition. A tree uses its spatial hash with a thread-safe hashmap to assess which trees are local to itself. Within this neighbourhood, trees within the tree's FON radius are considered for competition utilising Equation 1. If this tree's access to resources is considered to be dominated by another other plant, i.e. $I(a,b) = 0$, then the tree a is flagged as dead. It is then subsequently removed from the simulation in the next tick.

In addition, an initial job is scheduled prior to the simulation start. This job, `InitialSpawnTreesJob`, spawns an initial number of trees in the environment to start the simulation. Positions for these trees are pseudo-randomly selected within the world space. This step is carried out as forest growth is predicated on the existence of trees initially. Finally, a separate job named `UpdateForestJob` updates forest-wide simulation parameters, such as the wind direction and speed after each tick.

3.3. Spatial Hashing for Entity Neighbourhoods

A crucial bottleneck in Lane and Prusinciewicz' [LP*02] plant competition model concerns the individual competition between trees. Each tree in the simulation must determine which trees are within its FON radius, to assess if it is in competition for resources. The naive approach to this solution yields an algorithm with time complexity $\mathcal{O}(n^2)$, as each simulated tree potentially looks up every other tree in the simulation. In our solution, we utilise a uniform spatial hashing algorithm similar to Pozzer et al. [PdLPH14] for quantising an entity's position to a spatial grid index. An unmanaged `NativeParallelMultiHashMap` is used to provide a thread-safe concurrent map which entities can access to determine their local neighbourhood. It is worth noting that although virtual forestry typically occupies 3D space, our spatial hashing approach and simulation works in 2D; tree entities are simulated on the xy plane. This approach is taken as, firstly, it reduces the memory footprint of the simulation, namely the hashmap and ECS components. Secondly, this type of simulation has been used throughout the majority of forest simulation approaches [LP*02, AD15, WRH19]. It is worth noting however that this step is a simplification of real tree growth; restricting the environment to 2D could potentially impact the accuracy of the growth model. For example, previous work has shown that the altitude of trees can affect their growth and competition with other plants [CA07]. Examining this variable and its impacts on perceived quality could be another area of future research.

An entity's position $\mathbf{p} \in \mathbb{R}^2$ is hashed to a spatial index in two steps. Firstly, the grid delta value $\mathbf{g}$ is computed as $\mathbf{g} = \mathbf{w}/n$ where $\mathbf{w}$ is a 2D vector containing the world boundaries, and n is the number of subdivisions for the uniform grid. Then, the position $\mathbf{p}$ is hashed into a 2D index with $I(\mathbf{p})$ as seen in Equation 2.

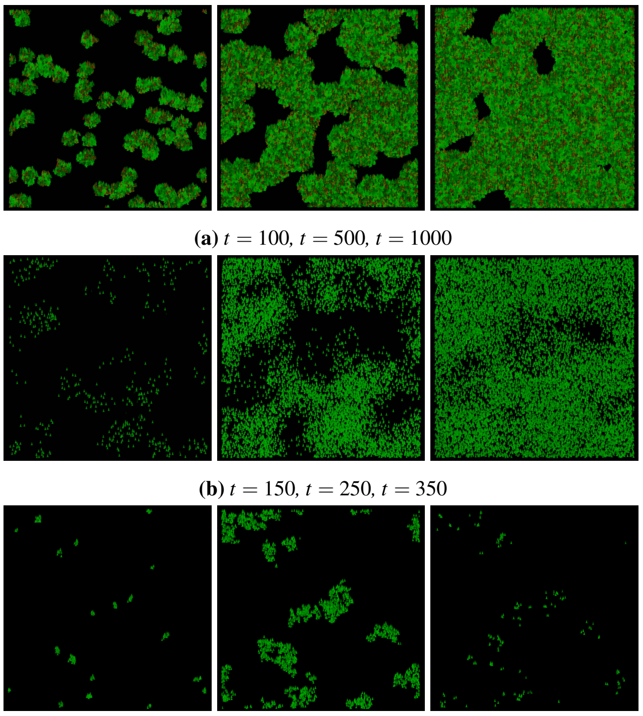
$$I(\mathbf{p}) = \left(\left\lfloor \frac{\mathbf{p_x}}{\mathbf{g_x}} \right\rfloor \quad \left\lfloor \frac{\mathbf{p_y}}{\mathbf{g_y}} \right\rfloor \right) \tag{2}$$

Finally, the 1D hashed index $\mathcal{H}(\mathbf{p})$ is calculated as $\mathcal{H}(\mathbf{p}) = I(\mathbf{p})_x + nI(\mathbf{p})_y$ where n is the number of grid subdivisions. Spatial indices are assigned through a parallel scheduled job every tick, as seen in Figure 4.

In addition, unmanaged components are utilised in our optimisation approach to represent the state of the simulation. Each `ForestComponent` instance contains forest-wide parameters used throughout simulation. For example, the minimum/maximum spread distance when plants propogate, the world size, wind direction, and so forth. In contrast, each `TreeComponent` represents an individual tree in the simulation along with its current state. A full list of data associated with individual simulated trees can be found in Table 1.

4. Results

Our optimisation strategy enables the real-time simulation of virtual forests with large ($n > 100,000$) entity counts. Trees are visualised with 3D meshes in a 2D distribution along the xy plane. Trees are oriented to $[0,0,1]^T$, the normal of the plane. Height is not considered in the visualisation process, though sampling according to a height-map may produce visually interesting results. In addition, colours can be assigned to the simulated trees: lighter shades indicate young trees, whereas darker colours indicate mature trees. Running the algorithm on a workstation with 32 GB of RAM, Intel Core i7-12700 processor and NVIDIA GeForce RTX 3080 GPU, the growth and rendering of the virtual forestry is performed in real-time ($\approx$ 60 frames per second). Some examples showcasing the simulation over time are shown in Figure 5. Parameters defining the plant competition can be tweaked to simulate forests of different shapes and sizes. The difference in parameterisation is emphasised when contrasting Figure 5a and Figure 5b. In Figure 5a, a small propogation distance and high spread chance is used to promote tightly clustered forests with clearings. However, in Figure 5b, the spread distance is heightened to create sparser distributions of trees. Finally, in Figure 5c, a small propogation probability is used. Plants are given a short period from maturity to death, giving a small window of opportunity to propogate within. This results in the emergent behaviour of periodic cycles of growth, similar to seasons. In addition, our optimisation strategy can be utilised in real-time photorealistic 3D scenes, as seen in Figure 6. The ability to simulate realistic forestry at run-time could be of particular use in video games and the development of virtual environments.


(a) $t = 100$, $t = 500$, $t = 1000$

(b) $t = 150$, $t = 250$, $t = 350$

(c) $t = 300$, $t = 400$, $t = 500$

Figure 5: *The simulation and development of three virtual forests at various time values t. The simulation in each case runs at approximately 60 FPS on a desktop machine, with a peak entity count of 249,771.*

5. Analysis

To examine the performance benefits of our optimisation strategy, a performance-based quantitative analysis was performed. In this study, the impact of spatial hashing and parallelisation on simulation frame time was considered in detail.

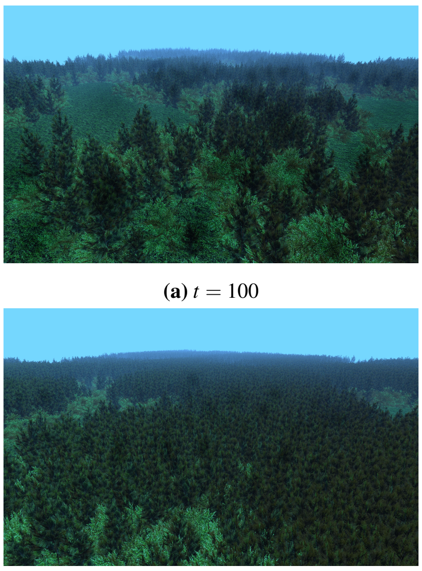
5.1. Spatial Hashing

To determine the role of spatial hashing in our optimisation strategy, the interframe latency time (delta time) was recorded and analysed. The number of spatial subdivisions for hashing was incrementally changed and simulated for 1,000 simulation ticks. For each run, the subdivision count was incremented by 1, in the inclusive range $[1,256]$. Following the 256th iteration, the application exited and data was serialised into a CSV format for further analysis. The delta time (time between each simulation tick, in seconds) was recorded and saved alongside the entity count and number of subdivisions. Following 1,000 ticks of a simulated run, the virtual forest was culled and reset. The pseudo-random number generator was additionally reset to an initial state, so subsequent forests would be identical in their seeding and growth. The simulation was run under this set-up a total of five times. Values from the five runs were averaged together to reduce data variance and strengthen the observed effects. Furthermore, the simulation was run on a desktop PC with 32 GB of RAM, a 12th generation Intel Core i7-12700 (2.10 GHz) processor, and NVIDIA GeForce RTX 3080 GPU. Jobs were scheduled in parallel as described earlier in this paper.

Our findings highlight the role of spatial hashing in drastically increasing performance for plant competition models. An example of this can be seen in Figure 7, in which the subdivision count is contrasted to the simulation tick time. As the plot shows, the frame render time is substantially lowered when the subdivision count g satisfies $g > 1$. This shows that spatial hashing, in-line with previous research [FWZS11], dramatically optimises processing time throughout simulation. It is worth noting that the optimal region of subdivisions for this particular simulation appears between the range $30 < g < 75$, and slowly increases as the subdivision count increases. However, in comparison to an approach which performs no local neighbour checks via a spatial hashmap ($g = 1$), perfor-

Table 1: *A detailed outline of each member of the `TreeComponent`, used in representing an individual tree's current simulated state.*

Member	Type	Description
`position`	`float2`	The position in the world of this tree.
`age`	`uint`	The age of the tree in the simulation.
`deathAge`	`uint`	The predetermined age at which this tree will die.
`matureAge`	`uint`	The predetermined age at which this tree will be able to propogate.
`needsCull`	`bool`	Whether the tree needs culling in the next tick.
`hash`	`int`	The 1D spatial index assigned to the entity.



(a) $t = 100$

(b) $t = 200$

Figure 6: *The simulation of a virtual forest in 3D space utilising our approach, at two time steps $t = 100$ and $t = 200$. Our approach is capable of generating and rendering 3D forest scenes at runtime.*

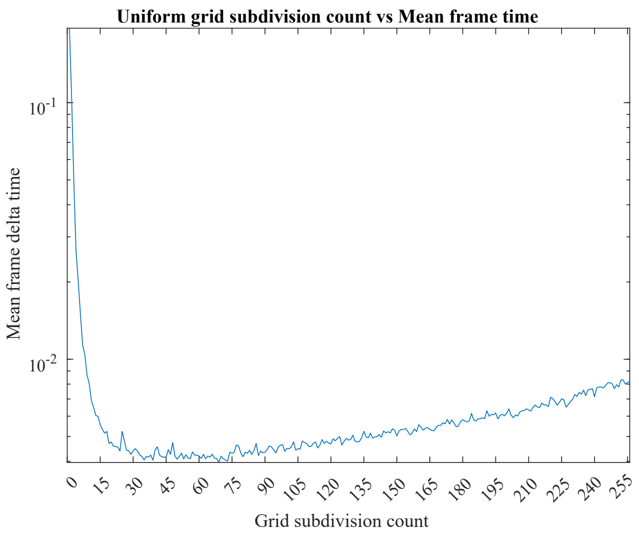


Figure 7: *A plot showing mean frame simulation time against subdivision count. Note that the y-axis in this plot is logarithmic.*

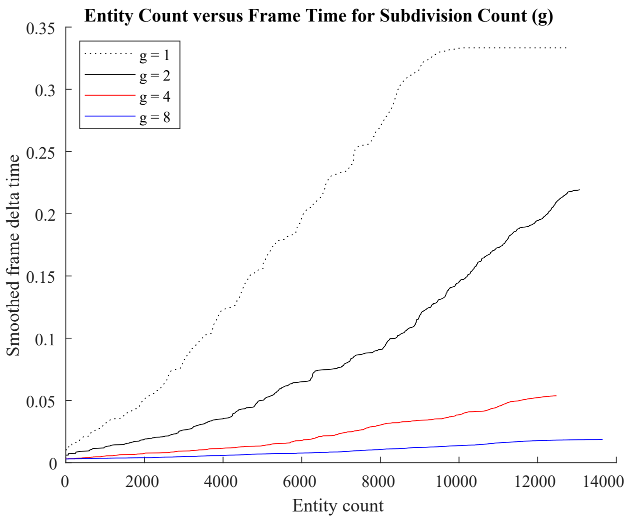


Figure 8: *A plot showing the relationship between entity count and recorded inter-frame latency, for a variety of grid subdivision counts g.*

mance is greatly increased when $g > 1$. Comparing the means of non-spatial hashing ($g = 1$, $\mu = .195, \sigma = .143$) against the most optimal spatial hashing run ($g = 73$, $\mu = .004, \sigma = .001$) shows a potential $.195/.004 = 48.75$ times reduction in frame delta time with spatial hashing. The difference is statistically significant between the two groups, $t(1998) = 42.734$, $p < .001$. A similar effect can be observed in Figure 8, with substantially lower frame processing times found for cases where spatial hashing ($g > 1$ in the plot) is utilised. Furthermore, a significant positive correlation can be seen between the number of simulated entities and processing time, $r = .41$, $p < .001$, $N = 8000$. The correlation between these two variables highlights the need for efficient real-time plant competition models – high numbers of entities are associated to high interframe latency. This is especially emphasised in the case of virtual forestry, where there may be hundreds of thousands of entities at any given time.

5.2. Parallelisation via Data-orientation

A similar analysis was performed to investigate the impact of utilising an ECS in our optimisation strategy. Unlike the previous analysis, two groups were used to determine the role of ECS in optimisation. Both groups did not utilise spatial-hashing throughout the simulation, to eliminate this as a variable in the recorded data. In the first group, the virtual forestry model was implemented using a serial approach, with no ECS or parallelisation via the job system. In the second group, the model leveraged the in-built ECS and made use of parallel job scheduling, as discussed earlier. The inter-frame processing time and entity count was recorded across 1,000 simulation ticks, similar to the previous analysis. This was carried a total of five times under both group conditions, and the recorded data was averaged together to reduce variance.

Some interesting results were found through the comparison between the serial and parallel versions, which can be seen in Figure 9. Firstly, it can be seen that the use of Unity's ECS system provides a dramatic increase in performance in comparison to a non-ECS equivalent. In the serial version, frame delta time rapidly increases throughout the simulation. This is likely due to the increased number of entities over time as trees propogate in the simulation. The plot does show an interesting effect however; for low numbers ($n < 100$) of simulated entities, there are marginal differences between a non-ECS and ECS approach. This indicates that

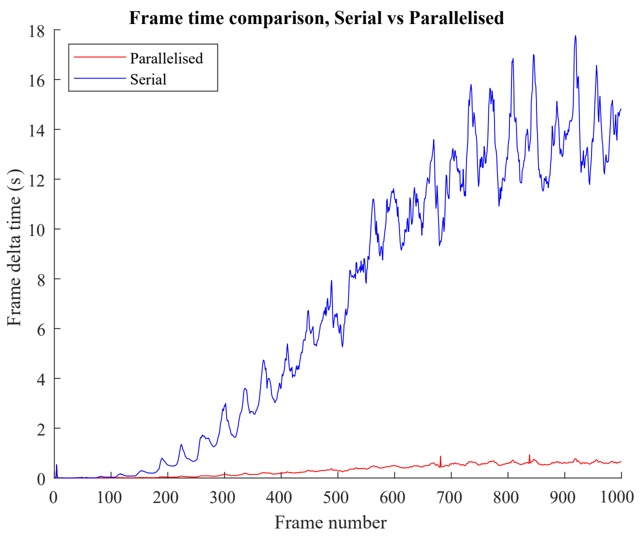


Figure 9: *A comparison between a serial (non-ECS) and parallel (ECS) implementation, for the first 1,000 frames of the simulation. Notice frame latency initially spikes and linearly increases for the serial version; whereas this is not observed for the parallelised implementation. Instead, the delta time for the parallelised version increases at a shallower gradient in comparison to the serial implementation.*

ECS may be useful for simulating large populations of entities, but unsuitable for smaller scale simulations. Comparison of the means between the non-ECS ($\mu = .7.057$, $\sigma = 5.534$) and ECS ($\mu = .321$, $\sigma = .241$) approaches shows a statistically significant difference, $t(1998) = -38.453$, $p < .001$. This effect indicates that the use of an ECS reduced frame delta time, on average, by approximately $7.057/.321 = 21.98$ times. Further work could investigate if this effect is present with the inclusion of a spatial hashing algorithm, i.e. $g > 1$. It could be the case that the inclusion of spatial hashing results in a different observed relationship between frame latency and simulation time. For example, the difference in latency between serial and parallel implementations could be lower when $g > 1$.

6. Conclusion

This paper has introduced a novel approach to significantly optimise FON-based plant competition models for real-time applications. In the approach, a combination of Unity's data-oriented technology stack, Unity's Entities package, uniform spatial hashing and job parallelisation are utilised to significantly decrease interframe simulation latency. The optimisation strategy discussed in this paper enables the efficient simulation of massive virtual forests for in-game scenes, a previously unsuitable task for real-time video games. In our findings, we show that uniform spatial hashing and the application of an ECS both dramatically increase simulation performance. Our parallelised version of a plant competition model outstrips leading serial implementations, providing an efficient tool for virtual forestry simulation. By utilising a spatial hashmap in tandem with an ECS, we found that simulation performance can increase by a factor of 48.75 against a naive $\mathcal{O}(n^2)$ algorithm. Parallelisation can increase performance by approximately 21.98 times, in comparison to a serial algorithm. Furthermore, the utilisation of Unity's data-oriented ECS, Burst compilation and job parallelisation additionally enhances simulation performance.

Additionally, we found that utilising an ECS provides most performance benefits at a large-scale with a large entity count. Leveraging an ECS for smaller-scale simulations leads to negligible benefits when contrasted to a serial implementation. This indicates that an ECS is a valuable tool for optimising computational tasks at scale. Our findings hope to inform the wider field of data-oriented Entity-Component systems, and their appropriate use in optimising large-scale simulations.

6.1. Future Work

In future work we wish to investigate other methods of optimising plant competition models. One such avenue may be through the utilisation of GPGPU, e.g. compute shaders, to provide further optimisation steps. We also intend to perform further rigorous analysis to investigate the role of ECS in optimisation. Examining, for example, the optimum thread pool size in parallel job scheduling, would provide a valuable information for games developers. Another interesting avenue could also consider finding an optimum subdivision count for spatial hashing, especially with regards to FON radius and world size. In addition, other spatial partioning algorithms, e.g. k-d tree partioning, may be another interesting direction for further research. It could be the case that k-d tree partitioning or a binary-space partioning algorithm may be more suitable for sparsely populated forestry. We are also excited to see the application of a data-oriented ECS in future work, especially in the optimisation of algorithms for real-time games. Another interesting direction for further research may also consider the impact of our strategy on memory performance, a task we have left for future work. Understanding the relationship between interframe latency and memory footprint within Unity's data-oriented framework, for example, could yield some interesting insights on the applicability of Entity-Component Systems for plant competition models. We plan to investigate the use of an ECS in further detail in future work.

Furthermore, one area left unconsidered in this paper is the integration of our strategy with LoD groups for tree instancing, or existing forest management SDKs such as SpeedTree's SDK. Building atop of this in future work would enable the seamless integration of our approach with the frontier of real-time rendering frameworks for virtual forestry. Finally, the open-source Unity project used throughout this paper is publicly available on GitHub, for further experimentation: `https://github.com/StaffsUniGames/pcg-forests-ecs`. We would like to invite contributors to help further the solution for future work.

References

[AD15] Alsweis M., Deussen O.: Procedural techniques for simulating the growth of plant leaves and adapting venation patterns. In *Proceedings of the 21st ACM symposium on virtual reality software and technology* (2015), pp. 95–101. 2, 3, 4, 5

[AK84] Aono M., Kunii T. L.: Botanical tree image generation. *IEEE computer graphics and applications 4*, 5 (1984), 10–34. 2

[Ber21] BERCHTOLD J.: *Pomegranate: Procedural 3D Tree Creation via User-Defined L-systems*. PhD thesis, California Polytechnic State University, 2021. 2

[BHRdA24] BADR A. S., HSIAO D. D., RUNDEL S., DE AMICIS R.: Leveraging data-driven and procedural methods for generating high-fidelity visualizations of real forests. *Environmental Modelling & Software 172* (2024), 105899. 3

[BLZ*11] BAO G., LI H., ZHANG X., CHE W., JAEGER M.: Realistic real-time rendering for large-scale forest scenes. In *2011 IEEE International Symposium on VR Innovation* (2011), IEEE, pp. 217–223. 3

[BN12] BRUNETON E., NEYRET F.: Real-time realistic rendering and lighting of forests. In *Computer graphics forum* (2012), vol. 31, Wiley Online Library, pp. 373–382. 3

[BWB*04] BAUER S., WYSZOMIRSKI T., BERGER U., HILDENBRANDT H., GRIMM V.: Asymmetric competition as a natural outcome of neighbour interactions among plants: results from the field-of-neighbourhood modelling approach. *Plant Ecology 170* (2004), 135–145. 2

[CA07] COOMES D. A., ALLEN R. B.: Effects of size, competition and altitude on tree growth. *Journal of Ecology 95*, 5 (2007), 1084–1097. 5

[Car19] CAREY B.: *Procedural Forest Generation with L-System Instancing*. PhD thesis, Bournemouth University, 2019. 3

[CGB*09] COURNÈDE P.-H., GUYARD T., BAYOL B., GRIFFON S., DE COLIGNY F., BORIANNE P., JAEGER M., DE REFFYE P.: A forest growth simulator based on functional-structural modelling of individual trees. In *2009 Third International Symposium on Plant Growth Modeling, Simulation, Visualization and Applications* (2009), IEEE, pp. 34–41. 3

[DeJ22] DEJONG T.: Simulating fruit tree growth, structure, and physiology using l-systems. *Crop Science 62*, 6 (2022), 2091–2106. 2

[DLY*20] DONG J., LIU J., YAO K., CHANTLER M., QI L., YU H., JIAN M.: Survey of procedural methods for two-dimensional texture generation. *Sensors 20*, 4 (2020), 1135. 2

[DRHK*21] DE REFFYE P., HU B., KANG M., LETORT V., JAEGER M.: Two decades of research with the greenlab model in agronomy. *Annals of botany 127*, 3 (2021), 281–295. 3

[ENMGC19] ECORMIER-NOCCA P., MEMARI P., GAIN J., CANI M.-P.: Accurate synthesis of multi-class disk distributions. In *Computer Graphics Forum* (2019), vol. 38, Wiley Online Library, pp. 157–168. 2

[FJFJ17] FAVORSKAYA M. N., JAIN L. C., FAVORSKAYA M. N., JAIN L. C.: Software tools for terrain and forest modelling. *Handbook on Advances in Remote Sensing and Geographic Information Systems: Paradigms and Applications in Forest Landscape Modeling* (2017), 69–109. 3

[FS18] FORD E. D., SORRENSEN K. A.: Theory and models of inter-plant competition as a spatial process. In *Individual-based models and approaches in ecology*. Chapman and Hall/CRC, 2018, pp. 363–407. 2

[FWZS11] FAN W., WANG B., ZHOU J., SUN J.: Parallel spatial hashing for collision detection of deformable surfaces. In *2011 12th International Conference on Computer-Aided Design and Computer Graphics* (2011), IEEE, pp. 288–295. 6

[GFK21] GARIFULLIN A., FROLOV V., KHLUPINA A.: Approximate instancing for modeling plant ecosystems. In *Proceedings of the 31st International Conference on Computer Graphics and Vision* (2021), vol. 31, pp. 95–104. 3

[HDRZ*03] HU B.-G., DE REFFYE P., ZHAO X., KANG M., ET AL.: Greenlab: A new methodology towards plant functional-structural model–structural part. In *Plant growth modelling and applications* (2003), TsingHua University Press and Springer, pp. 21–35. 3

[HMVDVI13] HENDRIKX M., MEIJER S., VAN DER VELDEN J., IOSUP A.: Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 9*, 1 (2013), 1–22. 1

[KCR*23] KUMARAN V., CARPENTER D., ROWE J., MOTT B., LESTER J.: End-to-end procedural level generation in educational games with natural language instruction. In *2023 IEEE Conference on Games (CoG)* (2023), IEEE, pp. 1–8. 2

[KGM14] KENWOOD J., GAIN J., MARAIS P.: Efficient procedural generation of forests. *Journal of WSCG 22* (2014). 3

[KS15] KOHEK Š., STRNAD D.: Interactive synthesis of self-organizing tree models on the gpu. *Computing 97* (2015), 145–169. 3

[KS18] KOHEK Š., STRNAD D.: Interactive large-scale procedural forest construction and visualization based on particle flow simulation. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library, pp. 389–402. 3

[LD05] LAGAE A., DUTRÉ P.: A procedural object distribution function. *ACM transactions on graphics (TOG) 24*, 4 (2005), 1442–1461. 2

[LKL22] LIEB S. J., KLEE N., LAWONN K.: Clasping trees-a pipeline for interactive procedural tree generation. In *VMV* (2022), pp. 49–56. 2

[LLB23] LEE J. J., LI B., BENES B.: Latent l-systems: Transformer-based tree generator. *ACM Transactions on Graphics 43*, 1 (2023), 1–16. 2

[LP*02] LANE B., PRUSINKIEWICZ P., ET AL.: Generating spatial distributions for multilevel models of plant communities. In *Graphics interface* (2002), vol. 2002, Citeseer, pp. 69–87. 3, 5

[LWW10] LIPP M., WONKA P., WIMMER M.: Parallel generation of multiple l-systems. *Computers & Graphics 34*, 5 (2010), 585–593. 3

[NFP22] NUNES R., FERREIRA J. F., PEIXOTO P.: Procedural generation of synthetic forest environments to train machine learning algorithms. In *ICRA 2022 Workshop in Innovation in Forestry Robotics: Research and Industry Adoption* (2022). 3

[NZ22] NEWLANDS C., ZAUNER K.-P.: Procedural generation and rendering of realistic, navigable forest environments: An open-source tool. *arXiv preprint arXiv:2208.01471* (2022). 3

[PdLPH14] POZZER C. T., DE LARA PAHINS C. A., HELDAL I.: A hash table construction algorithm for spatial hashing based on linear memory. In *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology* (2014), pp. 1–4. 5

[PH13] PRUSINKIEWICZ P., HANAN J.: *Lindenmayer systems, fractals, and plants*, vol. 79. Springer Science & Business Media, 2013. 2

[Pru86] PRUSINKIEWICZ P.: Graphical applications of l-systems. In *Proceedings of graphics interface* (1986), vol. 86, pp. 247–253. 2

[RB85] REEVES W. T., BLAU R.: Approximate and probabilistic algorithms for shading and rendering structured particle systems. *ACM siggraph computer graphics 19*, 3 (1985), 313–322. 2, 3

[spe] Speedtree documentation: Forest library. `https://docs.speedtree.com/doku.php?id=culling_population_structure`. Accessed: 2024-08-14. 3

[Ste23] STEINER M.: *Sketch based L-systems for tree modeling in virtual reality*. PhD thesis, Technische Universität Wien, 2023. 2

[TTWZ20] TANVEER M. H., THOMAS A., WU X., ZHU H.: Simulate forest trees by integrating l-system and 3d cad files. In *2020 3rd International Conference on Information and Computer Technologies (ICICT)* (2020), IEEE, pp. 91–95. 2

[VRP22] VENKATARAMANAN A., RICHARD A., PRADALIER C.: A data driven approach to generate realistic 3d tree barks. *Graphical Models 123* (2022), 101166. 2

[WH17] WILLIAMS B., HEADLEAND C. J.: A time-line approach for the generation of simulated settlements. In *2017 International Conference on Cyberworlds (CW)* (2017), IEEE, pp. 134–141. 2

[WRH19] WILLIAMS B., RITSOS P. D., HEADLEAND C. J.: Evaluating models for virtual forestry generation and tree placement in games. In *CGVC* (2019), pp. 65–73. 2, 4, 5

[WRH20] WILLIAMS B., RITSOS P. D., HEADLEAND C.: Virtual forestry generation: Evaluating models for tree placement in games. *Computers 9*, 1 (2020), 20. 2