Tree Generation Algorithms in Computer Graphics

Catarina Teves Martins da Costa
pg52676@alunos.uminho.pt

***Abstract*—Tree generation algorithms have been a main area in the computer graphics field due to their ability to generate a variety of tree outputs from a set of input parameters, allowing for realistic placement of trees in virtual scenes. This essay explores the complexity of tree generation and its algorithms, focusing on L-systems and DOL-systems, Space colonization, procedural modeling of trees along other techniques.**

***Index Terms*—Procedural modeling, generation trees, L-systems, tree reconstruction, space colonization**

I. Introduction

Trees, as we know, serve as invaluable resources for a multitude of purposes, ranging from providing habitat for wildlife and oxygen to offering essential ecosystem services.

However, in the computer science realm, tree generation and its algorithms have emerged as a powerful tool for simulating the intricate growth and development of trees.

This field encompasses a wide array of techniques, ranging from procedural methods that mimic natural growth patterns to more stylized approaches that cater innovative space colonization algorithms, L-systems and DOL-systems.

This ability to generate realistic or abstract tree structures has profound impact across various domains, such as video games, film production, landscape design, and environmental modeling.

As the renowned artist and designer *Bruno Munari* once said, *"The beauty of a tree lies in its complexity,"* . This highlights the intricate balance between simplicity and detail that characterizes this fascinating domain within computer graphics.

This essay will explore **L-systems and DOL-systems**, focusing on their foundational principles, applications, **Space colonization Algorithms** and its way of generating complex organic structures, **Procedural modeling** techniques using guiding vectors, and other relevant topics.

Fig. 1. Different species of trees generated with procedural model

II. Procedural Modeling

Trees belong to various species, each possessing a wide range of shapes corresponding to its specific type. Tree modeling in computer graphics encompass three primary categories: reconstructing from existing real-world data, interactive modeling methods, and procedural or rule-based systems.

Procedural methods allow for dynamic, environment-sensitive trees but suffer from low controllable due to the manual setup of input parameters, especially with the complexity required for modern large-scale scenes.

Besides that, in this section we will explore a type of procedural modeling that uses guiding vectors to build and generate trees.

A. Procedural Tree Modeling with Guiding Vectors

This algorithm of procedural modeling aims to only model the tree's architecture (spatial arrangement of trunk and branches) and not other details such as leaves and bark. The algorithm is automated, requiring no kind of user guidance or external data like point cloud or photographic representations of measured trees. Also, it is controlled by a collection of parameters, many of which can be left at default settings.

The algorithm is composed by four unique steps:

1) **Initial Population:** Populate with nodes the volume where the tree will reside, and link nearby nodes with directed edges.
2) **Minimum Cost Path Calculation:** Apply Dijkstra's algorithm to find the shortest paths from the source to all nodes. Edge weights are calculated lazily, setting a node's guiding vector upon opening as a rotation from its parent's vector, and assigning undefined weights based on the edge vector's relation to the guiding vector. Branches naturally align with guiding vectors due to lower costs along their directions.
3) **Endpoint Selection and Branch Structure Formation:** Select endpoints in the graph and determine the least-cost paths from the source to these endpoints. The union of all paths will form a branching structure that will eventually become the tree.
4) **Hierarchical Construction:** The source is updated to the set of paths obtained so far. Steps two and three are repeated, creating a hierarchical structure.

1) Key elements:

As mentioned, one of the key elements of this technique is the usage of **guiding vectors** to enhance graph-based tree synthesis, where each node's guiding vector influences the weight of outgoing edges. This encourages paths to align with the guiding vector field.

It is important to mention that in this algorithm, a guiding vector for a newly opened node is an incremental rotation of its parent's vector, rotated about an axis perpendicular to both the up direction and the traversed edge.

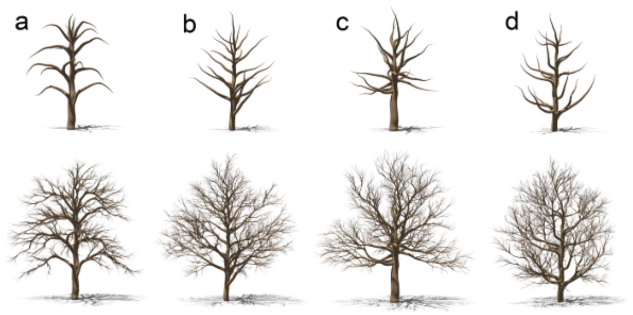

Fig. 2. Trees generated using different rotation settings. Above: results after two iterations; below: results after five iterations

Another key element is the **Early endpoint placement** since it significantly impacts the tree's overall shape.

Different initial placements lead to distinct final trees after several iterations, unlike synthetic topiary's clipping mechanism, this method produces less structured and more organic shapes.

The usage of multiple endpoints initially results in shrub-like trees, while a single endpoint fosters traditional tree structures with a central trunk.

Late endpoint placement is a key element that helps setting the tree's general shape. Since balancing preservation of the initial form with the development of the early structure is crucial, later endpoints are placed on a surface extrapolated from the current tree, defined by nodes 'k' hops away, ordered outward from the root, and selecting a fraction 't' of the outermost nodes for new endpoints.

This key element, refined in conjunction with subsequent placement adjustments, can synthesize specific tree shapes, offering flexibility in tree modeling.

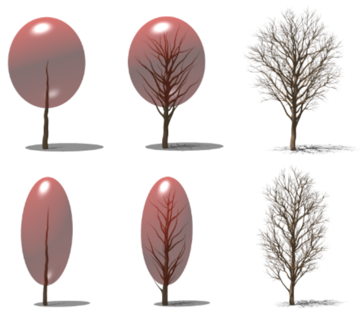

Fig. 3. Trees with trunks: bounding volumes control endpoint placement in the second iteration.

Lastly, the **Endpoint density** adjustments allows for the creation of sparser or denser synthetic trees, with models having more endpoints making a tree appear fuller.

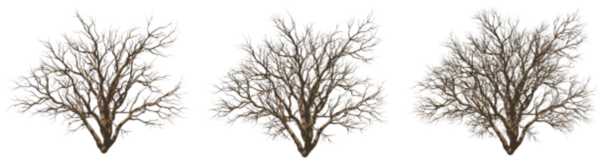

Fig. 4. Tree models with approximately 1000, 2000, and 4000 endpoints from left to right.

Overall, this technique enables the creation of detailed trees that mimic natural growth patterns, by influencing branch directions. It brings a more natural view into the Computer Graphics world.

However, it faces challenges when it comes to computational complexity and memory usage due to the need for multimodal adjustments and the potential need for higher resolution graphs.

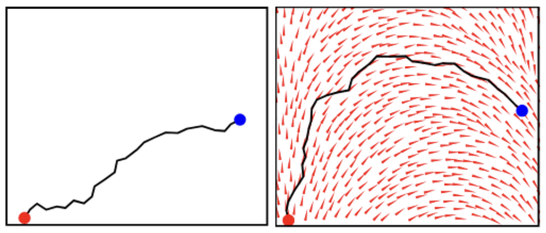

Fig. 5. Left: a path from graph without guiding vectors; right: a path from graph with guiding vectors.

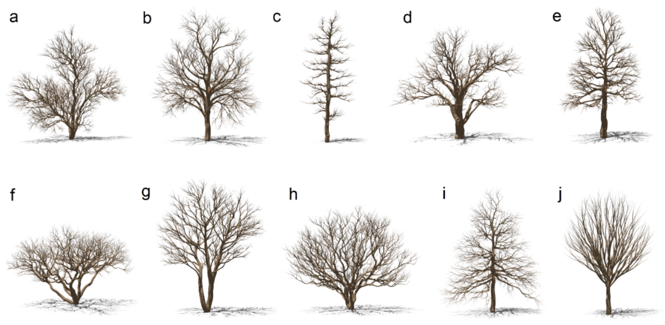

Fig. 6. A variety of different types of trees. using the tree model

These are some comparisons of the given tree model and other tree generation models.

Fig. 7. Left: a self-organizing tree model Right: guiding Vectors tree model.

Fig. 8. Left: a model from iterated graphs. Right: guiding Vectors tree model.

III. L-Systems

L-systems or Lindenmayer systems, are a type of formal grammar that emerged from the pioneering work of *Aristid Lindenmayer*, a biologist and botanist, in the late 1960s.

Initially designed to model the growth patterns of plants, L-systems have evolved into a versatile tool for generating complex structures in various fields, including computer graphics, architecture, and generative art.

L-systems are celebrated for their ability to generate self-similar fractals and have been adapted to model a wide array of phenomena, from herbaceous plants to neural networks and the procedural design of cities.

The central concept of L-Systems is rewriting. L-Systems consist of an alphabet of symbols used to:

- create strings
- an initial ”axiom”
- a set of production rules that expand each symbol into a larger string of symbols that generate them into geometric structures

The recursive nature of the L-system rules leads to self-similarity that make fractal-like forms are easy to describe.

This way plant models and natural-looking organic forms are easy to define in these type of systems, just by increasing the recursion level the form slowly ’grows’ and becomes more complex.

In tree generation, L-system can be used to describe trees as a way to create branches, just applying a set of rules, like the use of square brackets ’[ ]’.

For example, in the case of the ’[ ]’, anything placed within square brackets is treated as a separate instruction set executed by a new ”turtle” (a virtual pen that draws lines). This allows for the simulation of branches and the creation of more realistic models of plant growth.

It is important to mentions that these type systems can be moved to 3D scale just by applying a different set of rules.

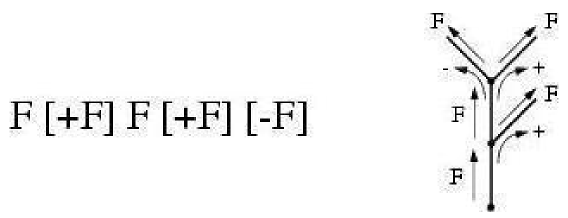

Fig. 9. Creation of branches in L-systems

There are different types of L-systems, among them we encounter the **DOL-systems** and **OL-systems**.

A. *DOL-systems*

DOL-systems are a simple class of L-systems, that are deterministic and context-free. They are fundamental for the generation of patterns and complex structures.

To better explain how DOL-systems work, lets consider the following example given by *Prusinkiewicz and Lindenmayer (1991)*:

Consider the strings ’a’ and ’b’, which can occur multiple times within a string. For each letter we specify a set of rewriting rules.

Rules:

- ’a’ is replaced by ’ab’ (’a $\rightarrow$ ab’)
- ’b’ is replaced by ’a’ (’b $\rightarrow$ a’)

The rewriting process begins with a specific string called the **axiom**, assumed here to be a single ’b’. Initially, ’b’ is replaced by ’a’ according to the rule ’b $\rightarrow$ a’. Next, ’a’ is replaced by ’ab’ following the rule ’a $\rightarrow$ ab’, resulting in the string ’ab’. Both ’a’ and ’b’ within ’ab’ are then simultaneously replaced, yielding ’aba’.

Following this pattern, ’aba’ evolves into ’abaab’, then ’abaababa’, and subsequently ’abaababaabaab’, continuing this process to generate increasingly complex strings.

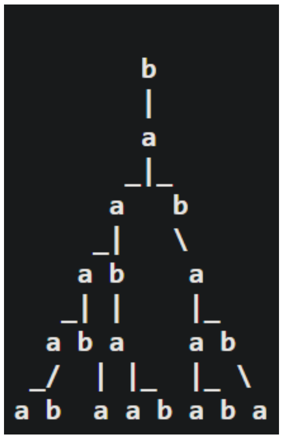

Fig. 10. Example of a derivation in a DOL-system

DOL-systems can also be used to emulate the development of multicellular filaments, just like the famous example of the Anabaena catenula.

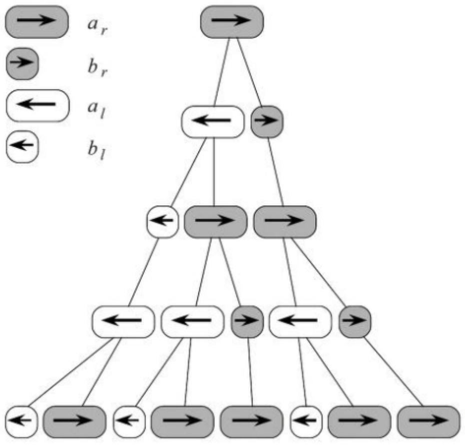

Fig. 11. Development of a filament (Anabaena catenula) simulated using a DOL-system. 'a' and 'b' represent cytological states of the cells

IV. Branching structures

Along this essay, it was mentioned that the plant kingdom is dominated by branching structures, therefore this section will discuss some methods to generate and manipulate these type of structures.

A. Axial trees

A rooted tree has edges that are directed and labeled. The edges sequences form paths from a distinguished node, called the root, to terminal nodes.

In a more biological sense, these edges represent branch segments, which are followed by at least one more segment in a path known as an internode. The terminal segment is referred to as the apex.

Axial trees are a special type of rooted tree. In each node, at most one outgoing straight segment is distinguished.

A sequence of segments is called an axis and together with all its descendants, an axis constitutes a branch, a branch is itself an axial (sub)tree.

In a axial trees, all branches and axes are ordered following a scheme for ordering branches introduced originally by Gravelius and further developed by Horton and Strahler.

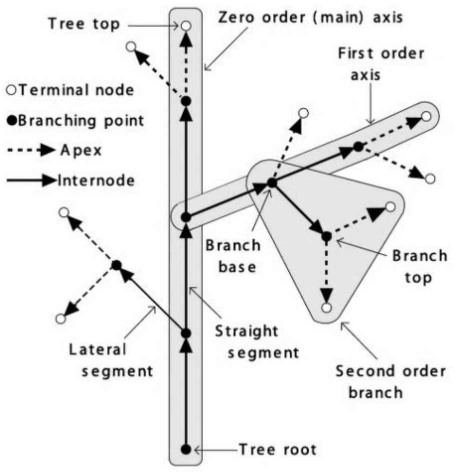

Fig. 12. Axial tree

B. OL-systems

1) Tree OL-systems:

Tree OL-systems, or parametric OL-systems, are an advanced subclass of L-systems capable of modeling branching structures and operating directly on axial trees. They employ a rewriting rule, or tree production, wherein a predecessor edge is replaced with a successor axial tree, aligning the start of the predecessor with the base of the successor and the end with the top.

A tree OL-system G includes three elements:

- a set of edge labels (V)
- an initial tree (w) labeled from V
- a set of tree productions (P)

2) Bracketed OL-systems:

Bracketed OL-systems, an extension of L-systems, incorporate a stack, a list of positions and directions, alongside two additional symbols, '[' and ']', to the alphabet.

The symbol '[' pushes the current position and direction onto the stack.

The ']' indicates moving the current position and direction to the top entry of the stack and remove that entry.

This mechanism allows for the direct manipulation of axial trees, enabling the creation of complex branching patterns by replacing predecessor edges with successor axial trees, aligning the start of the predecessor with the base of the successor and the end with the top.

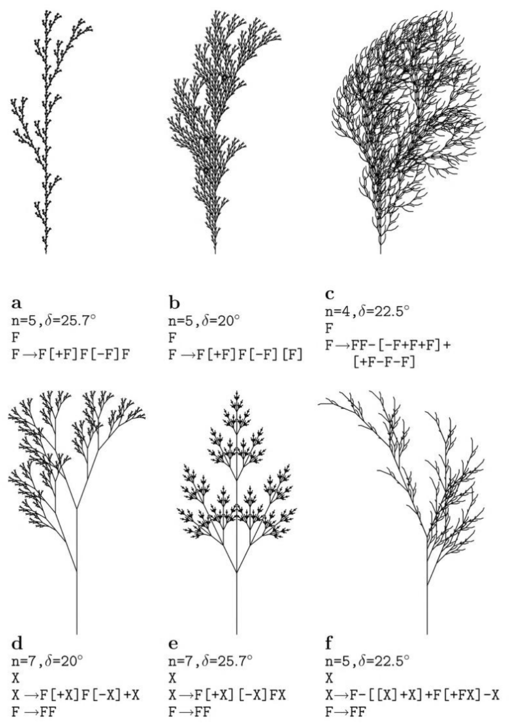

Fig. 13. Examples of plant-like structures generated by bracketed OLsystems. L-systems (a), (b) and (c) are edge-rewriting, while (d), (e) and (f) are node-rewriting.

Giving the examples showned in the images these are a good interpretation of the set of rules:

- 'F' represents the movement
- '+' right rotation by delta degrees
- '-' left rotation by delta degrees
- '[' pushes the current position and direction on top of stack
- ']' moves the current position and direction to the top entry of the stack

V. Other types of L-systems

A. Stochastic L-systems

Since all plants generated by the same deterministic L-system are identical, it results in a very artificial look to what nature really is. A way to prevent this is to use **Stochastic L-systems**.

Stochastic L-systems are an extension of deterministic L-systems, incorporating randomness. Unlike deterministic L-systems, where each symbol is replaced by a fixed string of symbols, stochastic L-systems allow multiple production rules for each symbol. In these systems, each rule has a certain probability of being chosen during each iteration.

Stochastic L-systems mainly affect the topology and the geometry of the plant, which makes it a powerful tool for creating new complex, organic-looking structures that mimic the unpredictability of Nature.

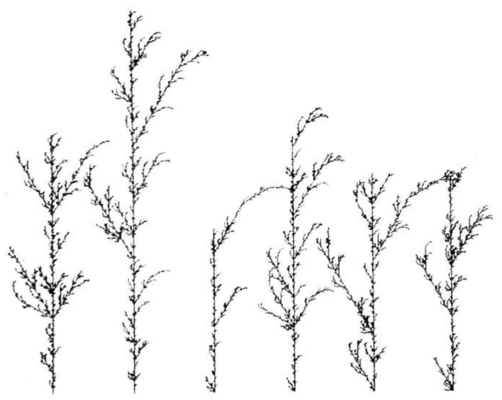

Fig. 14. Stochastic branching structures

B. Context-sensitive L-systems

Productions in OL-systems are context-free so they can be applied in everything regardless of the context in which the predecessor appears. However, production application may also depend on the predecessor's context. This effect is useful in simulating interactions between plant parts, like the flow of nutrients or hormones.

Context-sensitive L-systems incorporate the context of surrounding symbols into the production rules. This means that the replacement of a symbol depends not only on the symbol itself but also on its neighbors or the broader environment.

In deterministic context-sensitive, each production rule specifies the replacement of a symbol based on its immediate left and right neighbors. For example, a rule might state that a symbol 'X' should be replaced by 'Y' if it is flanked by 'A' on the left and 'B' on the right.

This type of L-system is extremely complex and comes with a lot of algorithmic and implementation challenges. It has been a case of study by Hogeweg and Hesper that listed 3,584 patterns, some of them obtained by type of L-systems.

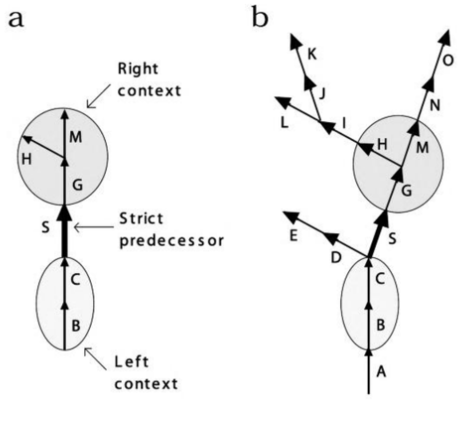

Fig. 15. Context-sensitive predecessor example

VI. Space Colonization algorithm

The **space colonization algorithm** is different from the other algorithms because is based on the concept of spatial competition, that plays the dominant role in determining the form of trees and shrubs.

In this algorithm a tree is formed in a natural, based-to-leaves order. All the parameters or attributes of the method are selected in the beginning of the process by the user. This provides control and makes consistent the characteristics of plant material used in landscaping, make it possible to generate a wide variety of forms.

These algorithms takes into account the following steps:

1) Place a set of attraction points, each one is associated with the tree node that is closest to.
2) *(The blue lines)* Figure out which attraction points are influencing which nodes.
3) *(The black arrows)* For each node, is calculated the average direction towards all of the attraction points influencing it.
4) *(The red arrows)* Calculate the positions of new nodes by normalizing the vectors, providing the basis for locating new tree nodes. Red circles
5) The new nodes are placed at the calculated positions
6) Check if any nodes are inside any attraction points *(Blue circles)* kill zones.
7) Removal of the attraction points that fit the criteria for the type of growth decided previously.
8) Begin the process over again from the step 2.

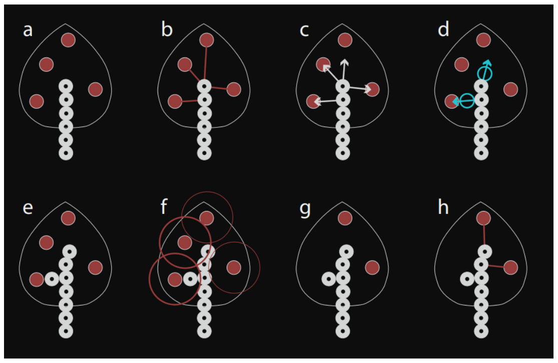

Fig. 16. The space colonization algorithm

As mentioned, the Space Colonization Algorithm initiates with a user-defined distribution of attraction points, which significantly influence the tree's final form. The entire procedure can be conceptualized as the following cycle:

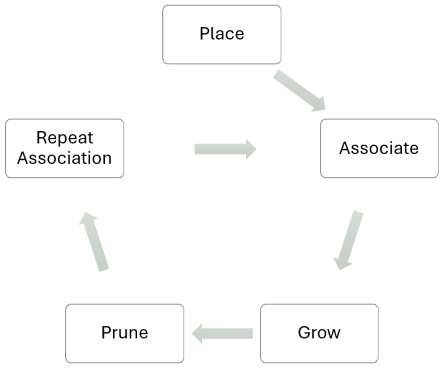

Fig. 17. The space colonization algorithm cycle

This algorithm draws inspiration from the observation of natural growth processes, particularly the iterative expansion of branching structures such as those found in leaves, trees, sea fans, and circulatory systems, like ant colonies.

It reveals great ability to produce realistic, organic-looking structures that mimic natural growth patterns and contrary to other algorithms it show a big adaptability, since its exploration strategies can be adjusted based on conditions encountered during execution, enabling to find optimal or approximate solutions for complex problems.

On the other hand, Space Colonization Algorithm requires careful consideration of the data structures and management strategies to efficiently handle the growth and branching logic.

VII. Others

Other techniques that are important to mention in the overall procedural generation of trees are the **Perlin Noise and Voronoi Diagrams**. Since they contribute to a more natural-looking vegetation.

A. Perlin Noise

Perlin Noise is a gradient noise developed by *Ken Perlin* in the 1980s. It generates smooth, continuous, pseudo-random values that can be used to simulate natural phenomena, such as the texture of tree bark or the undulations of a leaf. The algorithm operates in two, three, or four dimensions, involving three main steps:

1) Definition of a Grid of Random Gradient Vectors
2) Computing Dot Products
3) Interpolation Between Values

In tree generation, the perlin noise can be used to determine various tree variables like the height, width, and bending of tree branches.

By applying the noise to the coordinates of each branch segment, programmers can create branches that twist and turn organically, resembling the irregular patterns found in nature.

This technique is particularly useful for generating the intricate details of tree canopies and the varying thickness and angles of branches.

B. Voronoi Diagram

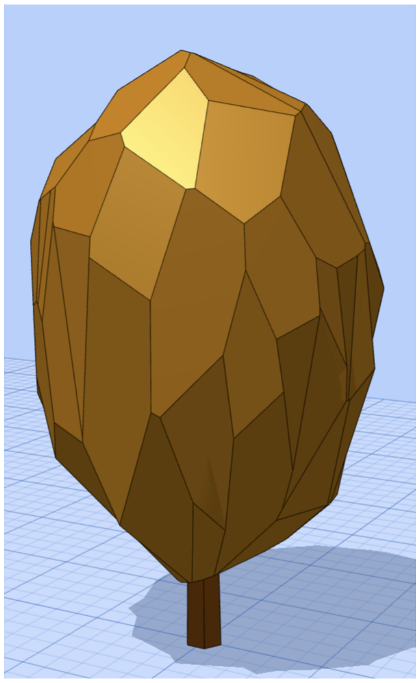

Fig. 18. Tree with the Voronoi Diagram

A **Voronoi diagram** is a type of tessellation pattern in which a number of points scattered on a plane. It divides a plane into regions based on based on distance to points in a specific subset of the plane.

For this technique, each point is assigned to the region closest to it. Voronoi diagrams pattern that they create can be found in nature, such as in cells and a giraffe's coat. They were are named after *Georgy Voronoy* and are widely used in computational geometry.

In tree generation, these type of diagrams can be employed to distribute tree seeds or nodes across a landscape in a way that mimics natural dispersion patterns. By placing seed points randomly and then drawing lines to connect each point to its nearest neighbors, it create a layout of trees that avoids overcrowding and maintains a sense of natural spacing.

The above image shows a tree generated with the usage of Voronoi diagrams.

VIII. Conclusion

Given the theme chosen to develop this article, algorithms, methods and techniques capable of generating trees were discussed and explored.

L-systems and **DOL-systems**, **Space colonization Algorithms**, **Procedural modeling techniques** using guiding vectors and other methods such as **Perlin noise** and **Voronoi Diagrams** had its place within the essay.

With that said, the generation of trees in computer graphics, using advanced algorithms is a very complex world that paves the way for the creation of realistic and interactive digital environments, in games, animations, urban planning and environmental simulations.

Although all the methods introduced were revolutionary, many problems still remain. One of them being the acceleration of computation, since it takes a long time to generate a realistic tree in some machines. Making the methods efficiency dependent on the technology limitations.

Either way these techniques combine mathematics, computer science and biology, offering a new perspective on the digital representation of nature in the computer graphics world.

Despite the challenges, there is still a great potential for innovation, as we continue to bridge the gap between the real and the computer worlds.

References

1) Mitra, N., Stam, J. and Xu, K. (2015). Pacific Graphics. [online] 34(7). Available at: https://gigl.scs.carleton.ca/sites/default/files/ling_xu/treemodelingwithguidingvectors.pdf
2) Dr.A.J.Marsh (2019). PD: Tree Generator. [online] Bitbucket.io. Available at: https://drajmarsh.bitbucket.io/tree3d.html.
3) Runions, A., Lane, B. and Prusinkiewicz, P. (2007). Modeling Trees with a Space Colonization Algorithm. [online] Eurographics Workshop on Natural Phenomena. Available at: http://algorithmicbotany.org/papers/colonization.egwnp2007.large.pdf.
4) Wikipedia. (2021). L-system. [online] Available at: https://en.wikipedia.org/wiki/L-system.
5) www-archiv.fdm.uni-hamburg.de. (n.d.). An Introduction to Lindenmayer Systems. [online] Available at: https://www-archiv.fdm.uni-hamburg.de/b-online/e28_3/lsys.html#BLS
6) www-archiv.fdm.uni-hamburg.de. (n.d.). An Introduction to Lindenmayer Systems. [online] Available at: https://www-archiv.fdm.uni-hamburg.de/b-online/e28_3/lsys.html#BLS
7) fay-barr (2014). PPT - Simulating Trees with Fractals and L-Systems PowerPoint Presentation - ID:6588547. [online] SlideServe. Available at: https://www.slideserve.com/fay-barr/simulating-trees-with-fractals-and-l-systems#google_vignette
8) GitHub. (n.d.). Build software better, together. [online] Available at: https://github.com/topics/space-colonization-algorithm
9) Webb, J. (2021). Modeling organic branching structures with the space colonization algorithm and JavaScript. [online] Medium. Available at: https://medium.com/@jason.webb/space-colonization-algorithm-in-javascript-6f683b743dc5.
10) Bellelli, F. (2023). The Fascinating World of Voronoi Diagrams — Built In. [online] builtin.com. Available at: https://builtin.com/data-science/voronoi-diagram.

11) rtouti.github.io. (n.d.). Perlin Noise: A Procedural Generation Algorithm. [online] Available at: https://rtouti.github.io/graphics/perlin-noise-algorithm.