Algorithms for procedural generation and display of trees

H. Nuić*, Ž. Mihajlović*
*University of Zagreb/ Faculty of Electrical Engineering and Computing, Zagreb, Croatia
hrvoje.nuic@fer.hr, zeljka.mihajlovic@fer.hr

***Abstract* – Main goal of this paper is to explore in which situations should some procedural algorithm be used to generate a tree model. Each algorithm is modified so as to be able to generate a tree model whose shape resembles the required model. Space colonization algorithm, algorithm using particle flows and algorithm simulating a Lindenmayer system are compared depending on a time needed to generate a tree with similar complexity. The voxelization procedure of the model and its application in this context is explained. Method for generating a tree mesh on a graphics card using Bézier's curves is presented.**

Keywords – tree; procedural generation; space colonization; L-system; particle flow; voxelization; Bézier curve

I. INTRODUCTION

Modelling and animating trees for movies, computer games and simulations are labour intensive work. Trees are most commonly represented as static objects with minimal interaction with its surroundings. Main reason is the sheer complexity of organic trees. Fully grown *common oak* can exceed the height of 40m and have more than 100,000 branches.

There are many commercially available tools which can help with modelling the tree, but the process is still time-consuming. Procedural algorithms can mitigate the time needed to create a desired tree model.

In this paper we have compared 3 algorithms. Every algorithm as an input gets voxelized model and as an output produces same abstract tree data structure. This data structure is further passed to graphics card which creates a mesh for every branch.

Procedural algorithms to some extent try to simulate the real growth of a tree, but as there are many unknown variables, it is hard to capture the real world. We have chosen to compare space colonisation algorithm, algorithm using particle flow and algorithm simulating a Lindenmayer system. Each one takes a different approach to the same problem. Space colonisation simulates a competition for space. Particle flow algorithm exploits similarities between branching structures in trees and trajectories of particle simulations. L-system focuses mainly on rules at which branching happens.

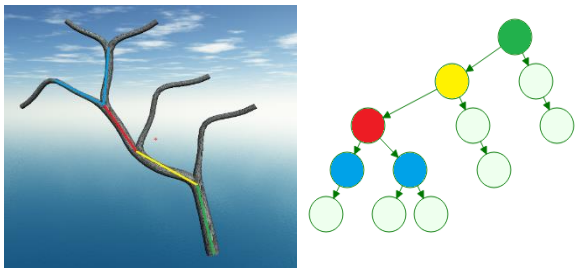

Figure 1. Rendered tree and its abstract tree structure

II. METHOD

In this section we will present the process of constructing a procedurally grown tree. Every algorithm is parametrized with a desired tree-crown size and voxelized model which we want to imitate.

Most common operation while generating a tree model with a desired shape is a test if some point is inside the model. While polygonal models for that operation have linear time complexity proportional to the size of a model, voxelized models have a constant time complexity. Main disadvantage of using voxelized models is a loss of precision, but with using 256^3 voxels and trees having irregular form there was no noticeable difference in a generated tree.

Models are voxelized with a variation of an algorithm using ray casting presented in a [1] and is described in the next section. Since the process of voxelization didn't take longer than a few seconds at the start of a program, there was no need for using a faster voxelization algorithm, like one presented in [2].

Every tree generator produces a tree data structure as demonstrated on a figure (1). From the perspective of a red branch, blue branch is its child, yellow branch is its parent, and green branch is its grandparent. Each node in a tree data structure stores its length and relative rotation to its parent branch. Main advantage with using abstract representation is that it reduces data that needs to be transferred to graphics card.

Abstract tree structure is converted on a graphics card to a mesh which is further rendered in a OpenGL graphics pipeline. This process is described in section VIII.

III. VOXELIZATION

To generate a voxelized model, we test if a centre point of each voxel is inside a model. Point is inside a model if the ray casted from the point intersects odd number of polygons, or in our case triangles. The ray we cast is always parallel to the Z-axis. That way, in the same time, we can test multiple voxels in the same column if they are inside a model. To speed up the algorithm, we project all polygons on a XY-plane and test which polygons affect which columns.

$$a_1 X_1 + a_2 X_2 + a_3 X_3 = X$$
$$a_1 + a_2 + a_3 = 1 \quad (1)$$

Barycentric coordinates a_1, a_2, a_3 are defined by the linear equations (1) described in a [3] where X_i are vertices of a triangle and X is a position of a point. Column of voxels is intersecting a triangle if its barycentric coordinates satisfy $0 \leq a_i \leq 1, i = 1, \ldots, 3$ condition. List of all affecting polygons is created for each column. Those lists are sorted by the height at which polygons intersect the column. That way, it is faster to count the number of intersected polygons for each voxel in a column.

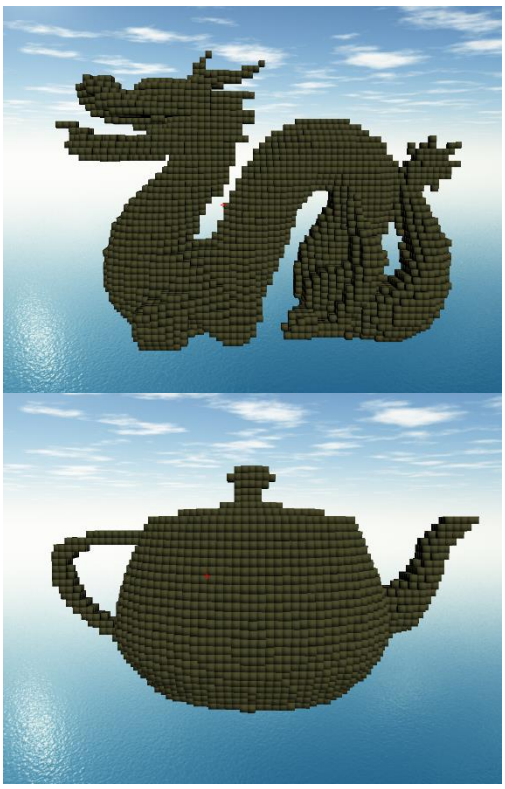

Figure 2. Voxelized models of the Stanford dragon and the Utah teapot

IV. SPACE COLONISATION ALGORITHM

Space colonisation algorithm has been introduced in a [4]. It was further expanded to 3D in [5] and used to construct a model of a tree. Algorithm has potential for other applications in different fields as presented in the [6] for creating a road network of a town and in the [7] for simulation of crowd dynamics. Algorithm is separated in two stages. Firstly, the model is filled with attraction points, then the algorithm creating branches is iteratively executed. Space defined by voxelized model is filled with attraction points. We have implemented linear distribution of attraction points in a voxelized model, but it is possible for example to arrange attraction points so that they are near the surface of a desired model. Resulting model would then have more branches near the surface.

We have implemented a variation of the space colonization algorithm using octree data structure presented in a [8], which has a complexity of an $O(m * n * \log(p))$, where n is number of iterations, m is number of finished branches and p is number of attraction points.

Every iteration consists of finding all attraction points close to every generated branch. Those attraction points are then written in a list for that branch. At the end of an iteration, new branches are created with the formula (2), where $\vec{n}$ is a vector representing a direction of a new branch, s is position of a current attraction point, v is the end position of current branch and S is a set of all attraction points affecting current branch.

$$\vec{n} = \sum_{s}^{S} \frac{s - v}{||s - v||} \quad (2)$$

$$l_b = l_{min} + \frac{l_{max} - l_{min}}{1 + e^{depth-10}} \quad (3)$$

Length of a branch directly affects the complexity of a generated tree, because smaller branches need more iterations to cross the same distance. Length of a new branch is calculated according to formula (3). l_{max} and l_{min} represent maximum and minimum desired length of a branch in a tree. Depth of a branch is the number of edges between a root branch and current branch in the abstract tree structure and is represented by variable $depth$.

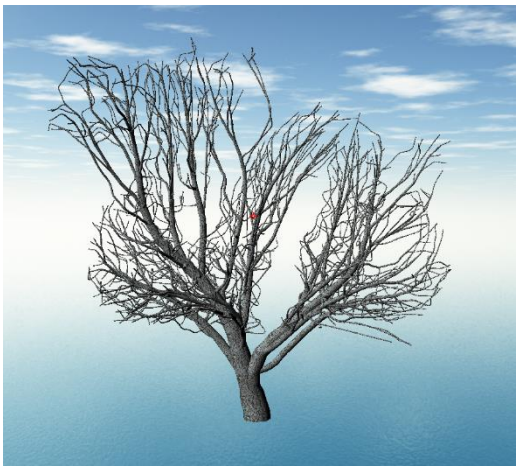

Figure 3. Stanford dragon created with space colonisation algorithm

V. ALGORITHM USING PARTICLE FLOW

We have implemented a variation of algorithm presented in [9]. It has been used in a [10] to generate trees that are imitating the branching structure from a photograph of a tree. By using image recognition and the help of human it is possible to generate a vector field which navigates particles so that their trajectories imitate a tree. Our implementation does not use vector fields and only forces that affect particles are global gravitational force and attraction forces between particles.

Algorithm using particle flow is also separated in two stages. Firstly, the space defined by the voxelized model is randomly filled with particles, then the positions of particles are iteratively calculated based on their last positions, masses and forces that affect them. Branches are constructed from trajectories that particles have passed through. New branch is created when the particle has passed minimum distance from the last created branch. Particles that are close enough are merged together and branches created by their trajectories are also merged.

$$\vec{F_i} = \sum_{j}^{N} k * m_i * m_j * \frac{x_j - x_i}{\left\| x_j - x_i \right\|^2} \tag{4}$$

Force on each particle is calculated with formula (4). Every particle affects every other particle with force proportional to their masses (m_i, m_j) and inversely proportional to their distance. Constant k is affecting overall force between particles. Positions of currently observed particles are x_i and x_j.

New velocities and positions of particles are calculated by the semi-implicit Euler method:

$$v_i = v_{i-1} + \Delta t * \frac{\vec{F_i}}{m_i} \tag{5}$$
$$\mathrm{x_i} = x_{i-1} + \Delta t * v_i$$

For simulation, it is necessary to define a Δt time step so that it is precise and fast enough. We chose 10ms as a step for generating a tree model.

When merging particles, it is possible to choose different combinations of particles which will be merged together. Greedy approach for every particle finds all particles that are close enough and instantly merges them. That approach introduces randomness in constructing a tree model, which suits the nature of the algorithm. Formulas (6) are used for merging particles in a bigger particle. Heavier particles influence more on a final position and velocity of a new particle. Variables $x, \vec{v}$ and m define position, velocity and mass of a new particle, while $x_i, \overrightarrow{v_i}$ and $m_i$define position, velocity and mass for merging particles.

$$x = \frac{\sum m_i * x_i}{\sum m_i}$$
$$\vec{v} = \frac{\sum m_i * \overrightarrow{v_i}}{\sum m_i} \tag{6}$$
$$m = \sum m_i * \overrightarrow{v_i}$$

VI. ALGORITHM SIMULATING A LINDENMAYER SYSTEM

Lindenmayer system is a type of a formal grammar which is suitable for describing fractal patterns in plants. It has been in use from 1968 and has been thoroughly studied and described in [11]. Visualized results obtained by L-systems convincingly simulate plants.

Like all formal grammars, L-system has a list of symbols V, starting symbol ω and production rules P. List of symbols can contain final symbols which cannot be further expanded. L-system is executed iteratively, and in every iteration, all active symbols are replaced with new symbols, with the help of a production rules. Result of executing a L-system is a list of symbols.

We have simulated the process of generating a tree by L-system. Algorithm can be executed until desired number of branches or iterations has been reached. Every symbol represents a branch in an abstract tree structure.

Every branch that doesn't have children are considered as an active branch. In every iteration one active branch is chosen which will produce its children. Number of children are calculated by the exponential distribution:

$$p(x|\lambda) = \lambda e^{-\lambda x} \tag{7}$$

Direction of growth for a new branch is calculated by addition of a random vector to the direction of growth of a parent branch. That way new branches are created without unnatural twists. If the direction of growth of a parent branch is parallel to the y-axis then figure (4) represents probability distribution of a new direction of a growth. Red colour represents most probable direction, and blue colour represents least probable direction. Newly created child branch is accepted to be active branch if it is inside the voxelized model.

Variations can be made to the algorithm by using different distributions for number of children, different distributions for direction of growth and different order in which active branches are selected for producing children.

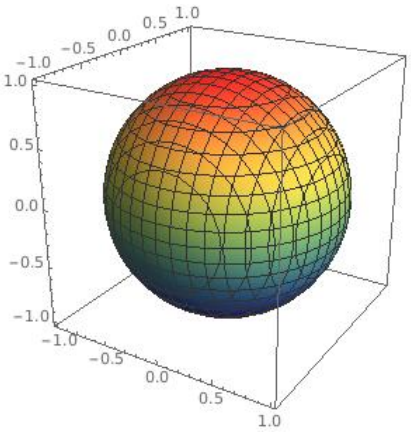

Figure 4. Probability distribution of a new branch direction

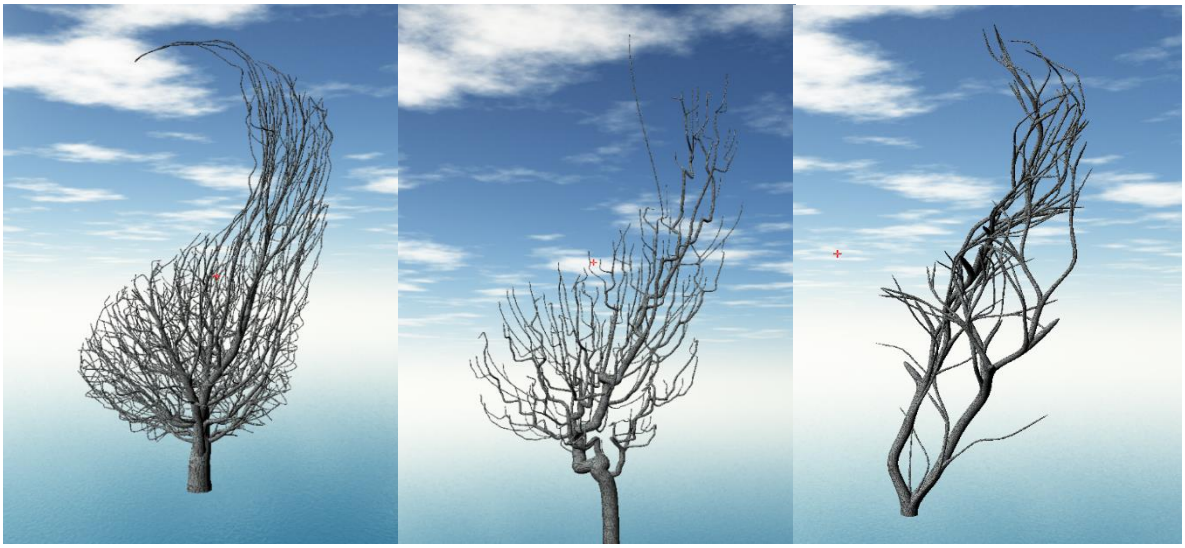

Figure 5. Resulting trees created with algorithms: space colonisation, particle flow and L-systems

VII. COMPARISON OF ALGORITHMS AND RESULTS

All three algorithms are successfully filling desired shapes and producing satisfying branching structures. Both space colonisation and particle flow algorithms can generate a tree structure that is not necessarily fully inside a desired model, but that problem can be solved by pruning the undesired branches.

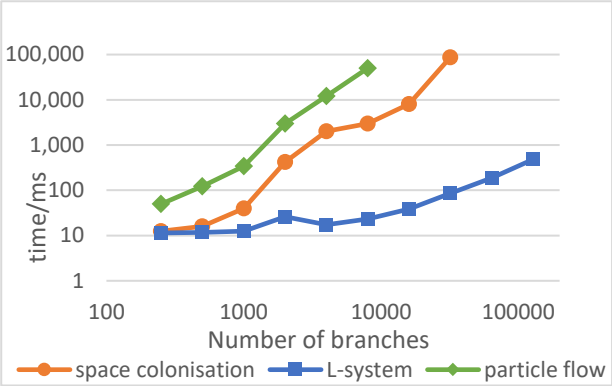

Figure 6. Time needed to create a desired number of branches

Figure (6) compares algorithms based on a speed at which they can create branches in a single tree. Both axes are on a logarithmic scale. Algorithms were tested on a single core of an AMD Ryzen 5 2400G with a clock rate of 3,6GHz.

Main advantage of using the L-system algorithm is its speed. It is possible to create many trees and branches in a short period of time, which is very useful for programs like flight simulator, where trees may need to be created in a real time. Algorithm is fast mainly because it doesn't care about already created branches. It has been recently used in a [12] with combination of adaptive level of detail algorithm to generate a forest in a short period of time. Another advantage is that it is easy to implement new rules which trees or other plants follow. L-system does not follow as good as other two algorithms a model we want to imitate with a tree, so it is not suitable for that task.

Main reason that particle flow is slower than other two algorithms is that it needs more iterations to generate a single branch. If time step in iteration is increased, then the simulation would not be precise enough and branches would pass more often one through another not detecting each other's presence. In comparison to space generation it can produce less intersections of branches.

Negative aspect of using particle flow is that it is hard to implement a new set of rules that would produce different types of trees. It has many parameters that need to be fine-tuned to get a satisfying result. Furthermore, as branches are created from leaves to the trunk, it is hard to predict on what depth will the branch be. That information would be useful to produce branches of different lengths depending on the position to other branches. Branches can be created after the simulation is over by storing data about every trajectory, but that way algorithm uses much more memory then needed. It is also possible to get too many branches connected to the same parent branch. For these reasons it is inferior in comparison to the other methods of generating tree models. Useful application of this algorithm would be for filling a tree model with leaves.

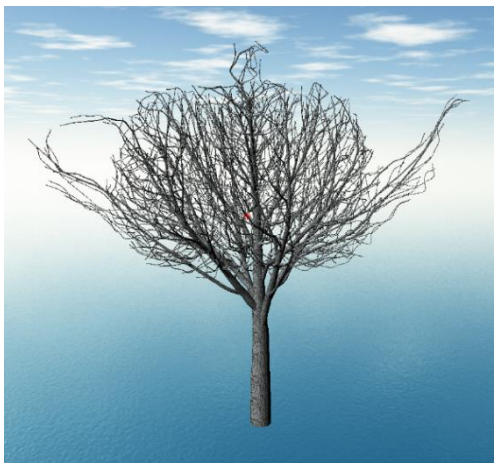

Figure 7. Space colonisation algorithm with the model of Utah teapot

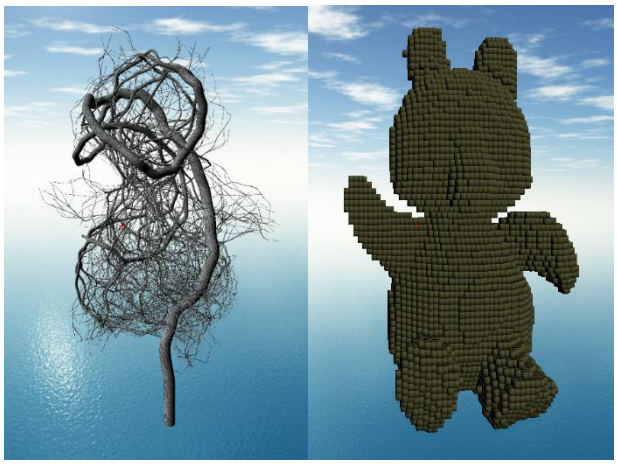

Figure 9. L-system algorithm with the model of a teddy bear

VIII. RENDERING THE TREE

Main goal of an algorithm for rendering a tree is to take an abstract tree data structure and convert it into a tree mesh. Algorithm can be executed on a CPU (central processing unit) or on a GPU (graphics processing unit). Advantage of running algorithm on a GPU is that we can dynamically change a shape and positions of branches in a tree while still maintaining a real time rendering. For average generated tree there was 0.5MB of data stored as a tree data structure. For that example, full mesh had 17MB of data. Main disadvantage of using GPU is that it is hard to do post processing of a tree mesh, like joining overlapping vertices of a branches which have the same parent branch.

Formula (8) from [13] is used to calculate radiuses of every branch, where x is exponent in a range of [1.8, 2.3] and it is dependent of a tree species we are trying to simulate. In our simulation, we used exponent of a 2.

$$r_{parent}^{x} = \sum_{every\ child} r_{child}^{x} \qquad (8)$$

To create a mesh of a branch, we need to know a radius of current and parent branches, furthermore we need to know positions of current, parent and grandparent branches. In total that is 11 floating point numbers. Creation of a tree mesh is done in a geometry shader in the OpenGL graphics pipeline.

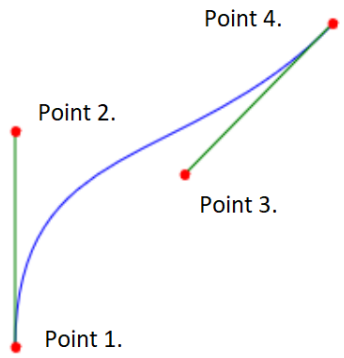

Figure 10. Cubic Bézier's curve used for rendering branches

Figure 11. Particle flow algorithm with the model of a bird

Main idea is to create a mesh that follows the Bézier's curve defined by the branches end points. Bézier's curve is defined by four points. First point is the end point of a parent branch. Second point is translated end point of a parent branch in direction of its growth. Third point is created by translation of a current branch end point in a negative direction of the current branch growth. Fourth point is the end point of a current branch. By choosing those points we get first derivation continuity between connecting branches.

To create a mesh around a branch we define vertices on a Bézier's curve in equal distances. Each vertex is the centre of a circle whose normal is parallel to the tangent of the Bézier's curve in that vertex. Radius of a circles is linearly interpolated between radius of a parent branch and a current branch. Newly created vertices are connected via triangle strip and passed further down the graphics pipeline.

Tree can be rendered statically and dynamically. Static rendering means that the data about positions of branches is not changed. Static rendering is fastest method of rendering a tree, because all data can be stored on a graphics card and it doesn't need to be changed. Dynamic rendering means that the tree structure and/or branch positions are changing in response to external stimuli. Dynamically

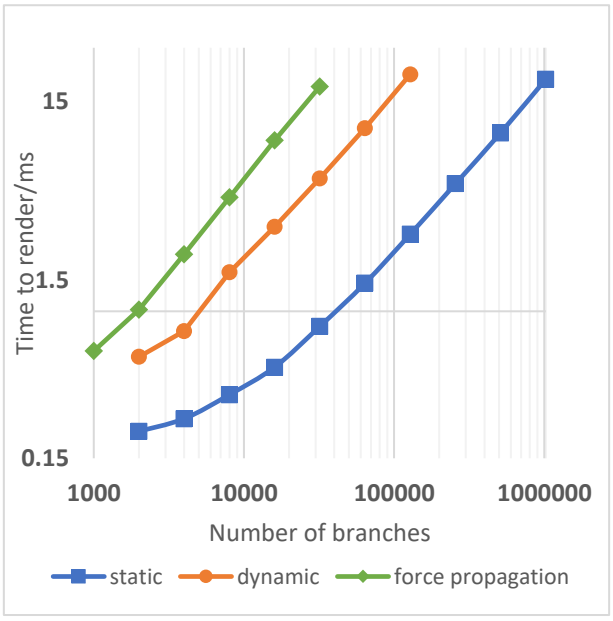

Figure 12. Comparison of different types of rendering

changing data about each branch, independently of other branches, costs us more processing power as seen on Figure (12). It would take even more processing power if we would like to simulate real world bending of branches in the presence of wind, with force propagation algorithm as presented in [14]. These rendering speeds are obtained on setup with a Titan V GPU.

IX. THE CONCLUSION

This work presents a variation to procedural generation algorithms so that they can generate trees with desired model shape. Their speed of generation and their positive and negative aspects are compared.

If it is necessary to create lots of trees in a short period of time, then it is best to use L-system algorithm. If we would like to create a tree with a desired shape, then the space colonisation and particle flow algorithms are much better suited. Models created with these algorithms are suited for simulations, movies and video games.

Voxelization of a model is explained. Results and speed of it are satisfactory. There are faster and better algorithms that parallelize voxelization but are more complex to implement.

Method for creating a tree mesh from an abstract tree data using Bézier's curve is presented. This method guarantees a first derivation continuity which produces meshes without unnatural twists.

ACKNOWLEDGMENT

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan V GPU used for this research.

REFERENCES

[1] J. Huang, R. Yagel, V. Filippov, and Y. Kurzion "An accurate method for voxelizing polygon meshes" In Proceedings of the 1998 IEEE symposium on Volume visualization, 119-126, 1998.

[2] M. Schwarz, and H. P. Seidel, "Fast parallel surface and solid voxelization on GPUs," ACM Transactions on Graphics (TOG), 29(6), 179, 2010.

[3] V. Skala, "Barycentric coordinates computation in homogeneous coordinates," Computers & Graphics, 32(1), 120-127, 2008.

[4] A. Runions, M. Fuhrer, B. Lane, P. Federl, A. G. Rolland-Lagan and P. Prusinkiewicz, "Modeling and visualization of leaf venation patterns" ACM Transactions on Graphics (TOG), 24(3), 702-711, 2005.

[5] A. Runions, B. Lane, and P. Prusinkiewicz, "Modeling Trees with a Space Colonization Algorithm," Eurographics Workshop on Natural Phenomena, 7, 63-70, 2007.

[6] G. D. Fernandes, and A. R. Fernandes, "Space Colonisation for Procedural Road Generation," In 2018 International Conference on Graphics and Interaction (ICGI), 1-8, 2018.

[7] A. de Lima Bicho, R. A. Rodrigues, S. R. Musse, C. R. Jung, M. Paravisi, and L. P. Magalhães, "Simulating crowds based on a space colonization algorithm," Computers & Graphics, 36(2), 70-79, 2012.

[8] D. Meagher, "Geometric modeling using octree encoding," Computer graphics and image processing, 19(2), 129-147, 1982.

[9] Y. Rodkaew, P. Chongstitvatana, S. Siripant, and P. Lursinsap, "Particle systems for plant modeling," Plant growth modeling and applications, 210-217, 2003.

[10] B. Neubert, T. Franken, and O. Deussen, "Approximate image-based tree-modeling using particle flows," In ACM Transactions on Graphics (TOG), 26(3), 88, 2007.

[11] P. Prusinkiewicz, and A. Lindenmayer, "The algorithmic beauty of plants," Springer Science & Business Media, 2012.

[12] Š. Kohek, and D. Strnad, "Interactive Large‐Scale Procedural Forest Construction and Visualization Based on Particle Flow Simulation," In Computer Graphics Forum, 37(1), 389-402, 2018.

[13] R. Minamino, and M. Tateno, "Tree branching: Leonardo da Vinci's rule versus biomechanical models," PloS one, 9(4), e93535, 2014.

[14] S. Pirk, T. Niese, T. Hädrich, B. Benes, and O. Deussen, "Windy trees: computing stress response for developmental tree models," ACM Transactions on Graphics (TOG), 33(6), 204, 2014.