Technical Report

forest_stand_generator_3D

v 0.1.4 (Python Package)

for Radiative Transfer Simulations

Date: January 14 , 2026

Author: Prasad Madushanka Dauglas Dambure Liyanage

1. Introduction

The **Forest Stand Generator 3D** (v 0.1.4) is a Python package designed to simulate realistic three-dimensional forest stands for ecological modeling and radiative transfer simulations. The software allows users to generate individual trees with customizable trunks and crown structures, place them in a forest plot using either uniform or random spacing, and visualize the results in both 3D and 2D projections. Leaf positions, orientations, and sizes are parametrically defined, enabling detailed analysis of canopy structure and light interception.

The package leverages **Python 3.10+**, **NumPy** for numerical computations, and **Plotly** for interactive visualizations. Users can export generated stands to **CSV** or **JSON** formats for downstream analysis or integration with other models. This tool is particularly useful for forestry, ecological, and environmental studies that require reproducible, parameter-driven forest stand simulations.

2. Architectural and Algorithemic Choices

The **Forest Stand Generator 3D** is designed with a modular architecture that separates tree modelling, stand generation, visualization, and data export, ensuring flexibility, maintainability, and clarity.

2.1 Module Architecture

- **tree.py –** Handles individual tree generation. Supports geometric crown shapes **(sphere, sphere_w_LH (upper hemi-sphere), cone, cylinder)**, leaf distribution, leaf radius sampling, and orientation via normal vectors.
- **stand.py –** Generates forest stands in rectangular plots with configurable spacing. Supports **uniform** or **random** placement and allows per-tree parameter variability.
- **visualization.py –** Renders **3D plots** of trunks and leaves and provides **2D top-down projections**. Leaf normals determine realistic orientation in 3D visualization.
- **export.py –** Facilitates exporting stand data to **CSV** or **JSON** formats, converting NumPy arrays to serializable forms.
- **data_validation.py** – Ensures that input parameters are valid and biologically meaningful.

2.2 Key Algorithmic Choices

- **Leaf Placement –** Leaves are randomly sampled within the geometric volume of the crown, ensuring that all leaves remain inside the crown boundaries. The sampling algorithm varies depending on the crown shape:
  - **Cone** – radius decreases linearly with height.
  - **Cylinder** – radius remains constant along the height.
  - **Sphere** – points are sampled inside a spherical volume.
  - **Sphere_w_LH** – points are sampled inside a spherical volume, restricted to the upper hemisphere only.

When crown shape is set to "sphere" and the crown height differs from the crown radius, the resulting crown geometry becomes ellipsoidal; similarly, when crown shape is "sphere_w_LH" and the crown height is not equal to the crown radius, an ellipsoidal upper hemispherical crown is generated.

- **Leaf Normals –** Leaf orientation vectors are crucial for realistic canopy representation. Implemented distributions:
    - **Uniform/ Spherical** – each vector is a random unit vector in 3D sphere space.
    - **Planophile** – fixed vector pointing along the horizontal plane (z-axis).
    - **Erectophile** – fixed vector pointing vertically (x-axis).

- **Leaf Area Index (LAI) –** Number of leaves is derived from LAI, crown radius, and mean leaf size. Ensures that the total leaf area matches the target LAI for radiative transfer studies.

$$N_{leaves} = \frac{LAI \times \pi r_{crown}^2}{\pi r_{leaf\ mean}^2}$$

- **Stand Placement Algorithms**
    - **Uniform** Placement – Trees are distributed evenly over the plot using a simple grid. Fast, predictable layout.
    - **Random** Placement with Minimum Spacing – Trees are placed randomly but checked for minimum distance to avoid overlapping trunks. Computationally expensive for large n due to pairwise distance checks.

3. Simplifications/ Assumptions

Several simplifications were intentionally made to keep the model simple, deterministic, and computationally efficient:

- Trees are simplified as a trunk and crown; fine details like branches or bark are ignored.
- Leaves are modelled as flat disks without thickness or curvature, with radii sampled from a normal distribution, and are randomly positioned within the crowns. Leaf colour is assumed to be uniformly green.
- When placing a tree in the stand, the trunk base centre is always located inside the rectangular plot, while the trunk radius is allowed to extend slightly beyond the plot boundary.
- Tree growth dynamics, and phenology are not considered.
- Terrain is assumed flat; environmental factors like wind, soil variation, ground vegetation or light competition are not included.

4. Limitations and Possible Extensions

- Crown shapes are limited to spheres, cones, cylinders, and sphere_w_LH; this can be extended to support more complex and realistic canopy shapes.
- The ground is assumed to be flat; slopes, terrain undulations, and obstacles are not considered. Uneven terrain models could be incorporated in future versions.

- The plot area is restricted to a rectangular shape; this could be extended to support arbitrary plot geometries and real locations using GIS vector processing libraries.
- Leaf color is fixed; future extensions could introduce a range of colors within a tree, including chlorophyll-based variability.
- Leaves are modeled as flat circular disks; more realistic leaf shapes with thickness and curvature could be considered.
- Branch structures are not modeled, and the trunk is assumed to be uniform; hierarchical branching systems with leaves attached to twigs and detailed trunk surface geometry could be added for improved realism.
- Currently, visualization is limited to 2D top and 3D views; future extensions could include side views and cross-sectional views to better analyze vertical canopy structure.
- Export functionality is limited to CSV and JSON formats; this could be extended to ecological data formats (e.g., LAS, SHP) and 3D mesh formats (e.g., OBJ, PLY).
- For large numbers of trees, memory usage and computation time can become significant; GPU-based leaf generation and parallel processing could be applied for large-scale scenes.

5. Computational Complexity

The computational complexity of the forest stand generator mainly depends on the **number of trees (n)** and the **average number of leaves per tree (m).**

- **Tree Generation :** Leaf generation dominates the cost, with each leaf sampled in constant time. Generating a single tree is **O(m)**, and generating the full stand is **O(n × m)** in both time and memory.
- **Stand Placement:** Uniform placement computes tree positions directly, resulting in **O(n)** complexity. Random placement with minimum spacing requires pairwise distance checks, leading to a worst-case complexity of **$O(n^2)$.**
- **Visualization:** Rendering trunks and leaves scales linearly with the total number of objects, resulting in **O(n × m)** time and memory complexity. Interactive 3D visualization can become a bottleneck for large scenes.
- **Data Export:** Exporting stand data to CSV or JSON involves iterating over all trees and leaves, resulting in **O(n × m)** time complexity with linear memory usage.
- **Overall Complexity:** Tree generation and visualization dominate runtime and memory usage, while random placement becomes expensive for large n. The implementation is suitable for small to medium forest stands but would require optimization, such as, spatial indexing or parallel processing for large-scale simulations.

6. AI Tool Usage

ChatGPT was utilized for generating Python docstrings, creating unit tests, converting algorithmic descriptions into executable code, and producing dummy datasets for forest stands consisting of 10–20 trees.

Adobe Firefly was used to generate the project package logo.