ChatGPT

Biological & Computational Models for 2D Plant/ Forest Growth

Various procedural models can simulate plant-like growth in a 2D side-scrolling world. **Lindenmayer systems (L-systems)** use rewriting rules (grammar) to generate branching structures. Originally developed to model plant cell topology [1], L-systems can produce highly organic-looking trees or ferns. (For example, *No Man's Sky* uses L-systems to generate alien flora [2].) A typical L-system draws branches via "turtle graphics," and can be extended with parameters or randomness for variety. Open-source tools exist – e.g. a Godot engine addon ("forest" repo) that draws trees from L-system rules [3]. **Strengths:** L-systems are fast and easy to implement, and yield high visual fidelity (see figure below). Rules can be hand-tuned or learned, and many libraries (even 2D sprite-based) support them. **Weaknesses:** The output is typically static once generated – the plant does not react to the environment or simulate resources. L-systems do not natively model roots, soil, or interactivity (unless extra rules are added).

Figure: Example 2D fractal tree generated by an L-system (see Solkoll's public-domain image). L-systems elegantly model branching geometry [1], but produce fixed structures after generation.

Recommendation: L-systems are a great first prototype for tiling worlds because of their low computational cost and rich visuals. They easily generate varied tree shapes with few lines of code. (For example, Terkwood's *forest* project implements Godot/Rust tree drawing from L-system rules [3].) Once growth rules are defined, drawing the plant is straightforward. If environmental interaction is needed (branches avoiding obstacles, light-seeking tips, etc.), the rules must be extended manually or combined with other techniques.

Reaction-Diffusion (Turing Patterns)

Reaction–diffusion models (e.g. Turing/Gray-Scott systems) simulate interacting chemicals that diffuse on a 2D grid, forming spots, stripes, or organic textures [4]. In principle one could use such models to generate grassy or forest patterns (e.g. patches of vegetation, leaf vein layouts, or moss patterns), but they rarely produce explicit branch structures. **Strengths:** RD systems naturally yield complex emergent patterns (like animal coat markings) from simple math [4]. They can create irregular, self-organizing textures that feel "natural." **Weaknesses:** They are computationally intensive (solving PDEs on each tile pixel) and tricky to control. Parameters must be tuned to get interesting results, and the output tends to be diffuse (spots/ stripes) rather than distinct trees. In a side-scroller, RD might be better used for background variations (e.g. ground cover or bark patterns) rather than simulating actual plant growth.

Cellular Automata (CA) Models

Cellular automata operate on a tile grid with simple local rules. CA models can emulate growth by letting "alive" plant cells expand or change state based on neighbors. For example, one Processing-based CA sim treats each plant cell as a grid cell and even simulates water flow, auxin and photosynthesis: new cells form

only if water/glucose levels suffice [5]. Another project grows semi-random 2D "tree" shapes: cells have probabilities to grow up or sideways depending on their height, mimicking sunlight-seeking branches [6]. In Liam Ilan's demo, cells are more likely to extend outward (horizontal) as they get taller, so a tall "trunk" branches out and avoids being overshadowed [6]. These CA can be combined with rules like "no growth if a neighbor above exists" to enforce one-sided growth.

*Figure: A CA-based plant simulation (Processing demo). Here each grid cell can spawn new cells (green leaves above ground, blue water, brown roots) based on rules (e.g. resource availability) [7] [8]. Cellular automata yield blocky but emergent plant shapes by iterating simple rules. *

Strengths: CA are well-suited to tile/pixel worlds. They naturally fit discrete grids and can model spreading phenomena (growth, decay, fire, diffusion) with local rules. The behavior is emergent: small rule changes can yield drastically different forest shapes. Many game devs already use CA for terrain or cave generation, and the same idea applies to plants. For example, the "Forest Generation CA" project grows forests in an interactive demo [8]. CA can also incorporate environmental feedback (light, water, competition) to make the world self-sustaining.

Weaknesses: CA shapes tend to look blocky or "pixel-art," which may limit realism. They require careful tuning of rules and probabilities (e.g. Liam Ilan's parameters [6]). The computation grows with grid size and number of iterations, which can be expensive for large worlds or many plants. CA also lack built-in continuous geometry, so smooth curved branches must be approximated by many tiles. Finally, CA typically need an initial seed or pattern and then propagate outward, so integrating CA plants into dynamic gameplay (e.g. reacting to player actions) can be complex.

Space Colonization (Branching Algorithm)

(Related to L-systems and CA) One powerful procedural tree algorithm is **space colonization**. It iteratively grows branches toward a set of "attractor" points (which can represent desired foliage locations or light sources). At each step, branch tips grow in the average direction of nearby attractors, then attractors that are reached or near a tip are removed [9]. This produces realistic branching networks and vein-like patterns. The space-colonization algorithm can be constrained by boundaries or obstacles [9], allowing trees to grow along uneven ground or around objects. It has been used in graphics (e.g. leaf venation models) and there are open implementations (see Jason Webb's 2D space-colonization experiments [10]).

Strengths: Space-colonization yields highly organic branching forms that can mimic real trees (dense foliage at attractors). It handles environmental constraints gracefully (you can define where growth "wants" to go). Compared to pure L-systems, it can adaptively adjust branch angles based on distributed attractors.

Weaknesses: It is still a primarily static generation method – once the tree is grown to its attractors, growth stops. It also requires generating or placing the attractor points (which could be random within a shape or based on light maps). Computationally, it's more expensive than a simple L-system rewrite (each step must find nearby attractors). But it can be done on a per-tree basis offline or in a background thread.

Agent-Based Models

In **agent-based simulations**, each plant or growth tip is treated as an autonomous agent that interacts with the environment (soil, light, other plants). For example, one could simulate each seedling as an agent that expands roots for nutrients or emits seeds based on local conditions. Research prototypes (often in ABM frameworks like NetLogo or Unity) have used this to study forest dynamics or root systems. The advantages are **rich emergent behavior**: agents can respond to local resources, compete with neighbors, or adapt over time, leading to complex ecosystem-level patterns. The downside is **computational cost and complexity**. Tracking thousands of plant agents (each with its own state and rules) is heavy. In a real-time game, this may be impractical unless heavily optimized. Most game implementations simplify agents – for instance, a 2D game might treat each tree as an agent that only checks collisions or does simple growth steps.

(For games, a hybrid is common: use L-systems or prefabs to draw each "plant agent," but have agents decide where *or* when *to sprout, spread seeds, or die.)*

Summary and Recommendations

- **L-systems:** Very fast and high-fidelity; excellent for generating varied tree/plant sprites or meshes quickly. Easy to tweak shape with rules. *Use first if you need static, detailed plants.* Many open projects exist (e.g. Godot L-system trees [3]).
- **Space Colonization:** Good for realistic branching; use as a second pass if you want more natural-looking trees influenced by environment.
- **Cellular Automata:** Offers dynamic, grid-based growth and decay. Medium cost; gives blocky but rich emergent forests (see the Processing CA examples [7] [8]). *Use next if you want plants that spread over time or respond to terrain.* CA can also simulate forest fires, regrowth, etc.
- **Reaction-Diffusion:** Very high cost; not typically used for explicit plant shapes. Could be used sparingly for texture or background pattern (e.g. grass/leaf distribution), but is **not** ideal for main tree growth.
- **Agent-Based:** Highest fidelity/emergence but also highest complexity. Best reserved for specialized sims or when every plant needs independent decision-making (e.g. a single large tree with wind/bending physics, or a detailed root network). For a side-scroller, full ABM is likely overkill unless heavily simplified.

Prototype order (considering cost and effect): Start with L-systems to quickly create recognizable plants (fast, easy, many examples [1] [2]). Next, experiment with a simple cellular automaton or rule-based growth for more dynamic forest behavior (e.g. Liam Ilan's forest CA [8]). If time allows, incorporate a space-colonization algorithm for any trees that need more natural shapes. Finally, use reaction-diffusion only if you want decorative organic textures, and agent-based modeling only if you need plants that strongly react or "learn" from the environment (not usually needed in 2D side-scrollers).

Sources: Algorithmic Botany (Prusinkiewicz) for L-systems theory [1]; examples of L-system and CA implementations [3] [8] [7]; No Man's Sky procedural flora [2]; Reaction–diffusion basics [4]; various demos and papers on emergent vegetation models [9] [5].

---

1 main.dvi
https://algorithmicbotany.org/papers/abop/abop-ch1.pdf

2 The algorithms of No Man’s Sky - Rambus
https://www.rambus.com/blogs/the-algorithms-of-no-mans-sky-2/

3 GitHub - Terkwood/forest: Draw trees using L Systems in godot and rust
https://github.com/Terkwood/forest

4 GitHub - jasonwebb/reaction-diffusion-playground: Interactive reaction-diffusion simulation with organic patterns and behaviors that emerge from the interactions of two chemicals mixed together.
https://github.com/jasonwebb/reaction-diffusion-playground

5 Cellular Automata Plant | Devpost
https://devpost.com/software/cellular-automata-plant

6 8 GitHub - liam-ilan/forest-generation-cellular-automata: A cellular automata for creating forests.
https://github.com/liam-ilan/forest-generation-cellular-automata

7 GitHub - JaryJay/cellular-automata: A plant growth simulation made with Processing.
https://github.com/JaryJay/cellular-automata

9 10 GitHub - jasonwebb/2d-space-colonization-experiments: Visual experiments exploring space colonization as a 2D morphogenesis tool.
https://github.com/jasonwebb/2d-space-colonization-experiments