Latent L-systems: Transformer-based Tree Generator

JAE JOONG LEE, BOSHENG LI, and BEDRICH BENES, Department of Computer Science, Purdue University, USA

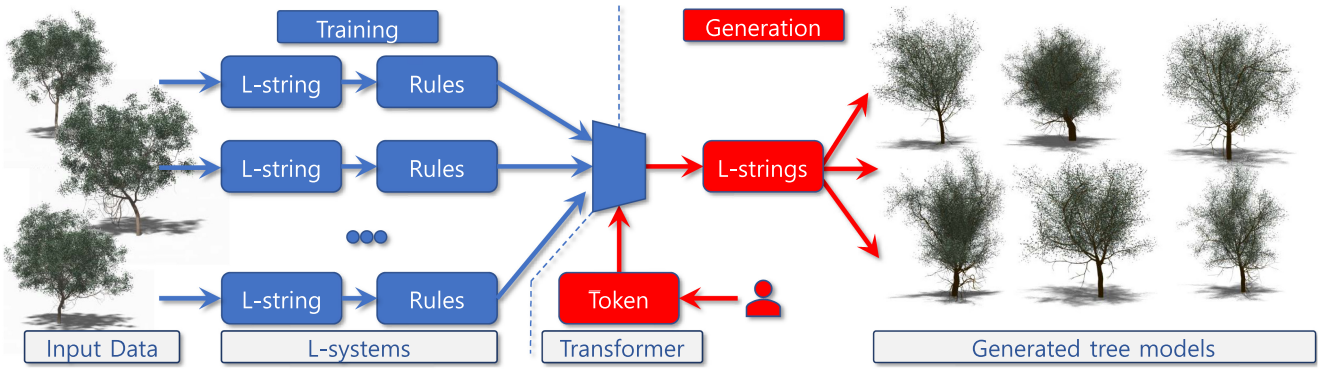

Fig. 1. We replace the tedious creation of L-systems with a deep neural model. A large set of 150k input 3D tree geometries is encoded as bracket-free L-system rules used to train a transformer. A new tree similar to the input is generated by running the transformer that produces an L-string that is converted to a 3D model.

We show how a Transformer can encode hierarchical tree-like string structures by introducing a new deep learning-based framework for generating 3D biological tree models represented as Lindenmayer system (L-system) strings. L-systems are string-rewriting procedural systems that encode tree topology and geometry. L-systems are efficient, but creating the production rules is one of the most critical problems precluding their usage in practice. We substitute the procedural rules creation with a deep neural model. Instead of writing the rules, we train a deep neural model that produces the output strings. We train our model on 155k tree geometries that are encoded as L-strings, de-parameterized, and converted to a hierarchy of linear sequences corresponding to branches. An end-to-end deep learning model with an attention mechanism then learns the distributions of geometric operations and branches from the input, effectively replacing the L-system rewriting rule generation. The trained deep model generates new L-strings representing 3D tree models in the same way L-systems do by providing the starting string. Our model allows for the generation of a wide variety of new trees, and the deep model agrees with the input by 93.7% in branching angles, 97.2% in branch lengths, and 92.3% in an extracted list of geometric features. We also validate the generated trees using perceptual metrics showing 97% agreement with input geometric models.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; • **Theory of computation** → *Rewrite systems;*

Additional Key Words and Phrases: L-systems, geometric modeling, neural networks

ACM Reference format:
Jae Joong Lee, Bosheng Li, and Bedrich Benes. 2023. Latent L-systems: Transformer-based Tree Generator. *ACM Trans. Graph.* 43, 1, Article 7 (November 2023), 16 pages.
https://doi.org/10.1145/3627101

1 INTRODUCTION

Getting a 3D geometric vegetation model has been an open problem in computer graphics for decades. Existing methods include reconstruction, manual authoring, and procedural (generative) models. Procedural algorithms provide a fully automatic generation of variations often needed in the industry. Still, procedural models are difficult to control, have non-intuitive sets of parameters, and often require an expert's knowledge. Recent generative approaches are getting close to simulators of biological behavior, and they suffer from an overwhelming number of control parameters. Yet, simple methods that would quickly provide many 3D tree models of real trees are not readily available.

Deep learning has been successfully used in various disciplines to generalize and compact input data. However, the application of deep learning for tree generation has been hindered by missing datasets and complex tree data representation. Deep neural models work well with regular structures such as images. Trees are branching structures with complex topology and geometry that include stochastic variations in branching angles, lengths of branches,

This research was supported by the Foundation for Food and Agriculture Research, United States Grant ID: 602757 to Benes. The content of this publication is solely the responsibility of the authors and does not necessarily represent the official views of the Foundation for Food and Agriculture Research. This work is based upon efforts supported by the PERSEUS award, #2023-68012-38992 under USDA NIFA to Benes. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NIFA or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.
Authors' address: J. J. Lee, B. Li, and B. Benes, Department of Computer Science, Purdue University, 305 N University St., West Lafayette 47907-2021, Indiana; e-mails: {lee2161, li2343, bbenes}@purdue.edu.

0730-0301/2023/11-ART7
https://doi.org/10.1145/3627101

curvature, the distance between branchings, and uneven data length.

The key inspiration for our work is two-fold. The first comes from recent advances in neural networks, particularly in Transformer [Vaswani et al. 2017] that has been shown to learn and translate sets of linear sequences. Our second inspiration is **Lindenmayer systems** (**L-systems**) [Lindenmayer 1968; Prusinkiewicz 1986; Prusinkiewicz and Lindenmayer 1990] that are an elegant and compact way of encoding trees as linear structures with hierarchies into so-called L-strings. We hypothesize that Transformers can be applied to encode hierarchies of strings represented as L-strings to learn tree topology and geometry and provide a compact and elegant representation that would generate large amounts of trees without tedious user control and L-system rules creation. The key problem to solve is that although L-systems are strings, they encode complex branching topology that precludes direct application of the Transformer to the problem.

We introduce a novel algorithm for procedural tree generation that generates valid L-systems for a wide variety of trees. Our method is an addition to the existing suite of procedural methods, but the main differences are that (1) it provides the complete description of the tree as L-systems strings and rules and (2) it is a data-driven method that generalizes the procedural models from existing datasets. In particular, our method uses a large dataset 155 k tree geometries of six species of trees generated by a developmental model [Li et al. 2021; Palubicki et al. 2009; Stava et al. 2014]. We convert the trees into a series of parameterized letters (L-strings) that encode the geometry as the terminal symbols and the tree topology (branching) as non-terminals. The key novelty of our approach is in converting each parameterized bracketed L-string into a hierarchy of strings for different levels of branching that are non-parametric and bracket-free. We then train six Transformers, one per species, that encode the geometry of each level by learning the distributions of terminal symbols and the topology by learning the distributions of the non-terminals. The neural networks generate new trees by initiating the trunk generator that outputs the first string that includes terminal and non-terminal symbols (branchings). Each generated non-terminal symbol then recursively executes the same Transformer to generate higher-level branches. The generation ends when all non-terminals are substituted. The output is a standard L-string that is then interpreted geometrically to generate the 3D geometry. Figure 1 shows several Acacia trees from 30 k samples used to train the Transformer.

Our work belongs to inverse procedural modeling. The key contribution is in replacing the tedious work of the manual creation of the L-system with a deep neural model that learns the procedural representation of existing trees. The generator is "aware" that it is generating, e.g., the trunk of a pine. It generates its structure by directly producing geometric terminal symbols, but it also generates the tree hierarchy by correctly positioning the non-terminal symbols indicating higher levels of hierarchy. An important property of this process is that it is all happening as string operations. Like L-systems, the strings are being rewritten, but instead of using the L-system rules, we use a deep neural network that has learned the rules from large amounts of data. We learn the implicit rules from the training data set and generalize them. Creating such rules manually is intractable. The second contribution is providing a lightweight algorithm with a small set of control parameters that can quickly generate the vast majority of complex 3D models. Our implementation takes, on average, 16 seconds to generate a tree geometry using a non-optimized Python generator. We will make the training dataset and the deep neural model publicly available upon acceptance.

2 RELATED WORK

Our work belongs to methods for vegetation geometry generation. We refer the reader to reviews about procedural generation of virtual worlds [Smelik et al. 2014], interactive vegetation modeling [Okabe et al. 2007], and tree modeling [Pirk et al. 2016].

Early **tree modeling** methods captured their fractal properties [Oppenheimer 1986; Smith 1984] and the tree structural repetitiveness [Weber and Penn 1995]. Still, the simple repetitive patterns cannot capture the environmental response that was the topic of the ongoing research, such as early methods based on particle systems [Aono and Kunii 1984; Arvo and Kirk 1988; Greene 1989]. Later methods were taking more inspiration from biology, such as the seminal article of de Reffye et al. [1988] and the interplay of the environment and the plant competition for resources [Palubicki et al. 2009]. The tree response to the environment was estimated from its geometry [Stava et al. 2014] and recent tree modeling methods introduce complex environmental effects such as support for obstacles [Hädrich et al. 2017], wind [Habel et al. 2009; Pirk et al. 2014], fire [Hädrich et al. 2021], deep neural models sensing the environment [Zhou et al. 2023], and roots [Li et al. 2023].

L-systems: One of the fundamental methods for the mathematical vegetation representation in computer graphics is L-systems (see Section 3 for detailed description). L-systems were inspired by linear cell subdivision [Lindenmayer 1968] and Prusinkiewicz [1986] introduced the geometrical interpretation that made them a commonly used model for vegetation simulation [Prusinkiewicz and Lindenmayer 1990]. L-systems were expanded in numerous ways, and among them, the most noteworthy is the ability to send signals between plant parts [Prusinkiewicz et al. 1993] and the modeling of the environmental response [Měch and Prusinkiewicz 1996]. One common problem of L-systems is their definition. The L-systems resemble a complex programming language, and, while advanced visualization APIs exist [Karwowski and Prusinkiewicz 2004], their creation requires advanced skills and a tedious trial-and-error process.

Inverse procedural modeling attempts to find a procedural description of an input object. Recent methods work of regular structures such as CAD models [Du et al. 2018], floorplans [Chen et al. 2019], urban models [Bokeloh et al. 2010; Vanegas et al. 2012], or facades [Wu et al. 2014]. Finding the procedural model for stochastic structures leads to approximate solutions. Inverse procedural modeling of trees has been addressed only in a limited way by Šťava et al. [2014] who estimate growth parameters of a developmental model and by a recently deep-learning approach that is capable of finding an L-system for 2D tree models from pixels [Guo et al. 2020]. Finding an L-system model for an existing 3D structure is an important open problem in the theory of string rewriting systems. Several approaches exist to the automatic construction of

L-systems, for example, by an alternative procedural modeling [Marvie et al. 2005] or by using genetic algorithms [Bernard and McQuillan 2021; Fitch et al. 2018; Ochoa 1998]. Close to our approach is the idea of generating programs to represent 3D models [Jones et al. 2020]. A statistical tree geometry generation has been used in [Wang et al. 2018], where a novel metric called Extended Square-Root Velocity Function generates trees statistically similar to the input dataset.

Contrary to previous work, we use a **deep neural generative model** that provides the output string. Deep learning has proven successful in generating new models from large datasets of examples (e.g., GAN [Goodfellow et al. 2014] for image generations, MaskGAN [Fedus et al. 2018] for text generations, MuseGAN [Dong et al. 2018] for music, polygonal meshes [Nash et al. 2020], variations of man-made objects such as chairs [Jones et al. 2020; Li et al. 2017; Mo et al. 2019], and deformable meshes [Gao et al. 2019]). Our work is closest to the methods that work with **Natural Language Processing (NLP)**, such as **Recurrent Neural Networks (RNNs)** [Li et al. 2018], GANs [Guo et al. 2018], and RNNs with the attention mechanism [Sutskever et al. 2014; Vaswani et al. 2017]. One of the recent achievements in processing sequential data using neural networks is Transformer based on the attention mechanism. The Transformer does not process the data in order due to the use of positional embedding. Moreover, it captures extended context information independent of past hidden states. Transformer shows a superb performance in language translations [Devlin et al. 2018] and is also the preferred method in our work. Transformer models have also proven effective in graph encoding (e.g., [Ying et al. 2021; Yun et al. 2019]) and have also been used to handle irregular structures in 3D space, such as proteins. PointBert [Brandes et al. 2022] shows the state-of-the-art performance in protein benchmarks as it encodes protein sequences into integer tokens. AlphaFold2 [Jumper et al. 2021] solves a problem to predict 3D protein structure by encoding the raw protein sequences, and the attention mechanism then prioritizes the influential information. Our representation is similar in that it encodes geometric information into sequences. The main difference is that we linearize the hierarchical tree structure and we encode them as L-system rules.

3 LINDENMAYER SYSTEMS

L-system [Lindenmayer 1968; Prusinkiewicz and Lindenmayer 1990] are a parallel string rewriting system described as a tuple

$$L = \langle M, \omega, R \rangle, \tag{1}$$

where $M = \{A, B, \ldots Z\}$ is a set of letters forming the L-system alphabet. Let M^+ denote a set of all non-empty sequences on M (reflective closure of the set) and let $M^* = M^+ \cup \emptyset$ denote all sequences (reflective-transitive closure). The symbol $\omega \in M^+$ denotes the L-system starting string (axiom). Finally, R is a set of production rules that define how a letter from the alphabet is rewritten. The rules have the form:

$$id_i : \quad A(p) : cond \quad \rightarrow x \in M^*. \tag{2}$$

Each letter of the alphabet A can have a set of parameters p that can be modified during production. A parameterized letter is called a module [Prusinkiewicz and Hanan 1990]. An example is $A(p) \rightarrow A(p/2)$, which halves the parameter's value. The condition *cond* is a boolean expression derived from the parameters. For example, the rule $A(p) : (p \geq 1) \rightarrow A(p/2)$ would halve the value of the parameter p only if it is greater or equal to one.

Table 1. Turtle Commands

Module	Turtle Command
F(d)	Move Forward by the distance d
+(α)	Turn Left by α
−(α)	Turn Right by α
\(α)	Roll Left by α
/(α)	Roll Right by α
&(α)	Pitch Down by α
^(α)	Pitch Up by α

String rewriting (the derivation) is an application of production rules in parallel on a string X, denoted by $\Rightarrow$, and it involves parallel rewriting of all letters in X according to R. The derivation starts with the axiom and creates a sequence $\omega \Rightarrow m_1 \Rightarrow m_2 \Rightarrow \ldots$. If the letter can be rewritten, i.e., there is a corresponding rule, it is called a non-terminal symbol. Letters that do not have rules are called terminal symbols. Terminal symbols are copied in the production. The derivation ends when there is no rule for the letters in m (i.e., $m_i = m_{i-1}$). In other words, the derivation ends when m includes only terminal symbols. If R includes recursive rules, it could run to infinity. In this case, the fixed number of rule applications is set explicitly. The size of R is the number of the production rules n and $i = 1, \ldots, n$. The key novelty of our approach is in substituting this process with a deep learning model.

The above-described L-system is a **deterministic zero context L-system (D0L)** because it always generates the same sequence. It can be extended to a stochastic L-system:

$$id_i : \ p_i : \quad A(p) : cond_i \quad \rightarrow x_i \in M^* \tag{3}$$

$$id_j : \ p_j : \quad A(p) : cond_j \quad \rightarrow x_j \in M^*. \tag{4}$$

Note that the same symbol $A(p)$ appears multiple times on the left-hand-side of the rules and each rule has additional probability p_i and p_j ($\sum p = 1$). A random choice is made determining if id_i or id_j will be used depending on the probabilities p_i and p_j.

Prusinkiewicz [1986] extended L-system in three ways. First, L-systems are interpreted geometrically by a LOGO-like turtle that reads the strings sequentially and interprets each module geometrically. Table 1 shows the symbols used in this article and their geometric interpretation. The ± symbols rotate the turtle left and right around the forward direction, the & and ^ rotate back and forward, and the pair of symbols \ and / roll the turtle around its main axis. The second extension is by defining the state of the turtle. The state $S = \langle P, \phi \rangle$ is its position P and orientation ϕ (one angle in 2D, two angles in 3D). The third extension adds special symbols represented by brackets (bracketed L-systems). When the turtle reads the symbol [, it pushes its status onto the stack. When it reads the closing bracket], it pops the status from the stack and sets its position to P and orientation to ϕ. The content of each bracket defines (recursively) a branch and bracketed L-systems have been used to mathematically describe trees, bushes, and other branching structures [Prusinkiewicz and Lindenmayer 1990].

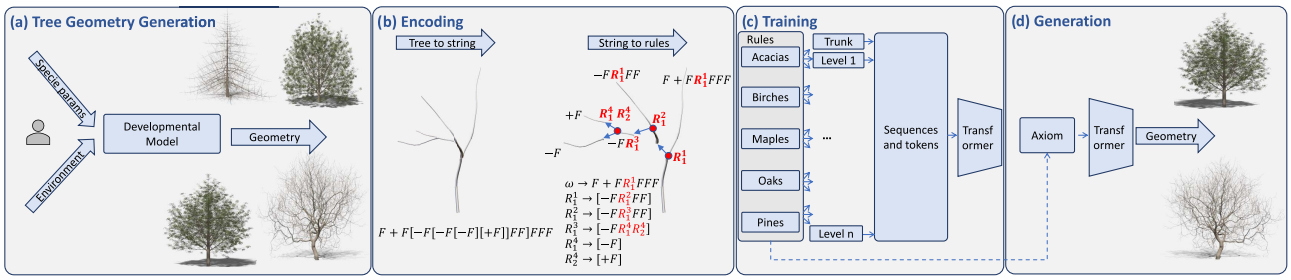

Fig. 2. Overview: The developmental model generates a large dataset of various tree species (a). The models are encoded as bracketed L-strings, deparameterized, and the hierarchies are flattened. The L-strings are then encoded as rules for each species and level of hierarchy (b). The rules are then converted to sequences of contextual tokens that train the Transformer (c) that can quickly generate new and unseen 3D tree models (d).

4 OVERVIEW

Our method consists of four steps (see Figure 2): (1) training tree geometry data generation, (2) encoding, (3) deep model training, and (4) new geometry generation.

The **tree data generation** step (Section 5) provides a large dataset of widely varying tree geometries by using a developmental (growth) model that models five tree species and environmental factors (gravity, light, and space colonization). The developmental model is complex and requires insight into the parameters and their meaning. The output of the first step is a large set of 3D geometries -we use 155,000 trees stored as 31.2 GB of data- represented as tree skeletons and 3D textured meshes.

We then **encode** the tree geometry for deep learning training, which is the key step of our proposed approach (Section 6). In the first step, the tree geometry is analyzed, branch order is determined, and the entire tree is encoded as a large sequence of letters (L-system string shown in Figure 2(b)). Some letters carry information about the tree geometry, and the topology is encoded by sequencing the letters using two special symbols: opening and closing brackets that denote tree branches (see Section 3 for details). We attempted to learn the L-string directly and use a recurrent neural network with an attention mechanism to generate new strings. However, the pairs of brackets that identify branching preclude such a naïve approach as it breaks the pairing of the generated brackets. Also, two parts of the tree that are physically close may be apart in the string representation because of the inserted branches. Instead, we convert the L-string into a sequence of bracket-free hierarchical rules, each corresponding to a different level of the tree hierarchy. This can be intuitively understood as cutting the tree branches and putting them in different categories depending on their hierarchy level. A branch of a higher hierarchy level in the string is represented by a non-terminal symbol indicating that a sequence should be used instead. This also brings close together the parts of the tree that are close in the tree geometry.

The **deep model training** (Section 7) uses the branches organized into hierarchies. While the branches are generated for each tree individually, the generated data is shared for all trees thus providing a large dataset of tree trunks, branches from the trunk, and so on. We designed our neural network with an attention mechanism and trained one network per species that keeps the information about pairing across the different levels. An important aspect is the distribution of the non-terminal symbols (topology) in the string that is efficiently learned by the network in the learning step.

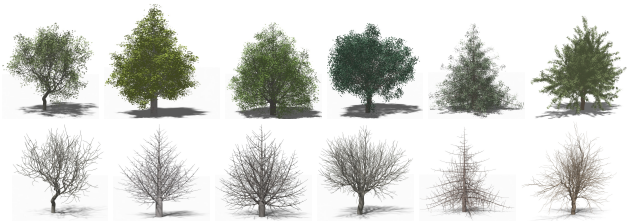

Fig. 3. Examples of the synthetic trees generated by our growth model and used as the training data with (top) and without leaves (from left: acacia, birch, maple, oak, pine, and walnut).

The new tree geometry is **generated** (Section 8) by providing a small starting sequence of letters (the axiom) to the network. In the same way that L-systems generate the geometry from the letters, we execute the model, generating a string of letters, including the non-terminal symbols that indicate branches. Each non-terminal carries information about the hierarchy level (small branchlets growing out of the trunk differ from the main branches). Then the same network is called for all non-terminal symbols, and the process continues until all non-terminals have been rewritten. The final string is then interpreted by the standard L-system mechanism (the turtle [Prusinkiewicz 1986]) that produces the 3D tree geometry.

5 TREE MODEL DATA GENERATION

We need a large dataset of 3D tree geometries to train our deep model. While real trees would be ideal, the state-of-the-art 3D tree reconstruction methods do not provide sufficient details, precision, and quantity required for deep learning. We are unaware of any dataset of real 3D trees that could be used. However, computer graphics tree geometry generation techniques are sufficiently mature to provide detailed 3D models. Although they require skilled users, they allow for a high level of control to generate large databases of 3D geometric models quickly. We use our tree geometry generator that is based on space colonization by competition for resources [Palubicki et al. 2009] and developmental and environmental response [Pirk et al. 2012; Stava et al. 2014]. We will provide the tree dataset for public use upon acceptance of the article.

Our tree generator supports the generation of Acacia, Birch, Maple, Oak, Pine, and Walnut (see Figure 3). The trees are de-

scribed by their botanical structure [De Reffye et al. 1988] such as the branching angle, phyllotaxis, and type of branching. The tree generator is a full developmental model based on De Reffye et al. [1988], Palubicki et al. [2009], Stava et al. [2014], Li et al. [2021] that models a plant bud behavior. The basic geometry is described by the structural biological model [De Reffye et al. 1988] that defines the direction in which each bud grows and how often they branch. The environmental response modifies the basic behavior by bending the growth direction against gravity (gravitropism) and towards the light (phototropism) [Měch and Prusinkiewicz 1996]. Buds also compete for space by space colonization that optimizes the branch 3D distribution to maximize light intake [Palubicki et al. 2009]. We also model internal signaling by simulation auxin (plant growth hormone), e.g., a bud that has been in a shadow for a prolonged time will become dormant and sends signals to the neighboring buds that will increase growth (light seeking).

We generated 155 k trees, 5 × 30 k per each species but 5 k for Walnut trees. (see Table 2 and Section 9.2). Leaves are generated automatically on branch tips, and tree branch width is generated by following the updated da Vinci rule [Minamino and Tateno 2014].

6 TREE ENCODING AND RULE GENERATION

The growth model provides 3D tree geometry where branches are represented as a set of discretized (piece-wise linear) generalized cylinders [Měch and Prusinkiewicz 1996]. We first convert the 3D geometry to a single string of L-system modules. Then we remove all the parameters from the modules by converting them to a string of letters, and we also remove the brackets by encoding the L-string to L-system rules. Eventually, we represent each string as a set of rules that are encoded as tokens for the Transformer.

6.1 Tree Ordering

We represent the tree as a skeleton that is a topological graph that also stores geometric information [Du et al. 2019]. In particular, each node stores data about the width, orientation, and a list of branches. The 3D mesh can be fully reconstructed from the skeleton.

It is unclear what constitutes a new branch and what is the continuation of an existing one. For example, a Y-shaped branching could be either a branch that splits into two (sympodial branching) that would lead to a string $F[+F][-F]$, or it could be a single branch (either left or right) with a new branch on the opposite direction leading to either $F[+F] - F$ or $F[-F] + F$ (monopodial branching). This cannot be disambiguated only by inspecting the local neighborhood because the branching could be only a small local branch. Holton [1994] introduced Gravelius ordering [Gravelius 1914] to geometric tree modeling, and we also adopted this approach. Gravelius ordering is a topological ordering of rivers that determines where the river starts and ends. It assigns order one to the river mouth and orders to the higher-order branches in a bottom-up fashion. The importance of each river at a confluence is determined and decides if the river is a continuation or a branching. A continuation inherits the order, and a new branch has a higher order (see Figure 4).

The tree skeleton is a directed graph $\mathcal{T} = [V, E]$, with nodes $V = \{v_i\}, i = 1, \ldots, n$ and edges $E = [v_i, v_j]$ oriented from the root

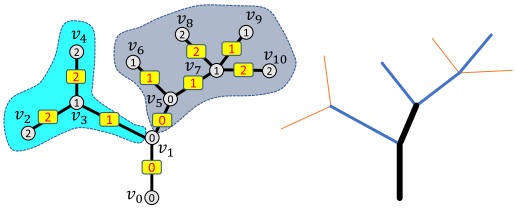

Fig. 4. Gravelius ordering assigns an order to each vertex and edge by determining if it is a continuation or a branching (left). The ordering defines consecutive branches (right).

to the leaves. The root v_0 is assigned its Gravelius order $g(v_0) = 0$. At each branching node v with branches v_i, the order of v is determined by weighting the depth of the subtrees and the branching angle. We use the following two conditions. The node v is a continuation if the depth of the subtrees $depth(v_i)$ does not vary by more than 20%. The second condition states that the branching angle should not exceed a certain value. The value is given by the average of the apical angle and the branching angle from the growth parameters for each species (Acacia: 30^o (average of 10^o and 50^o), Birch: 23^o (1^o and 45^o), Maple: 25^o (2^o and 48^o), Oak: 25^o (average of 20^o and 30^o), Pine: 40.5^o (1^o and 80^o), and Walnut: 30^o (25^o and 35^o)).

The order of an edge is set to the order of its end vertex $g(e) = g([v_i, v_k]) = g(v_k)$. An example in Figure 4 shows the Gravelius order of each node and edge as a yellow box. The depth difference of the two subtrees of v_1 is only one, and the right tree also has a small branching angle, so node v_1 is determined to be a continuation and has the Gravelius order of its predecessor $g(v_1) = g(v_0) = 0$. While the depth difference at node v_3 is also small, the branching angle makes the order $g(v_3)$ to increase to $g(v_3) = g(v_1) + 1$.

6.2 L-string Tree Encoding and De-Parameterization

The tree skeleton with the Gravelius ordering is parsed, and the string of L-system modules (parameterized letters from Section 3) is generated. We call this step *L-string tree encoding*.

L-string encoding: Each internode (a branch segment between two possible branchings) is converted to the $F(x)$ module, where x is the internode length (see Table 1), and each turning is represented as the corresponding command with the parameter storing the angle. There are various ways to encode branching as the following four strings represent the same geometry of the Greek letter Ψ:

$$(a)F[+F][-F][F] \quad (b)F[[+F] - F]F$$
$$(c)F[[[+F] - F]F] \quad (d)F[+F][-F]F,$$

and we use the last option (d), which minimizes the recursion level.

At each branching point, the order of children is verified. Each child of a higher order is processed in the following way: the push terminal symbol [is generated, the child node is recursively processed, and at the end, the pop symbol] is generated. If the branching has a child of the same order, the processing continues without generating the push and pop symbols.

L-string de-parameterization: The generated L-string is parameterized, and the parameters are interpreted as the value that

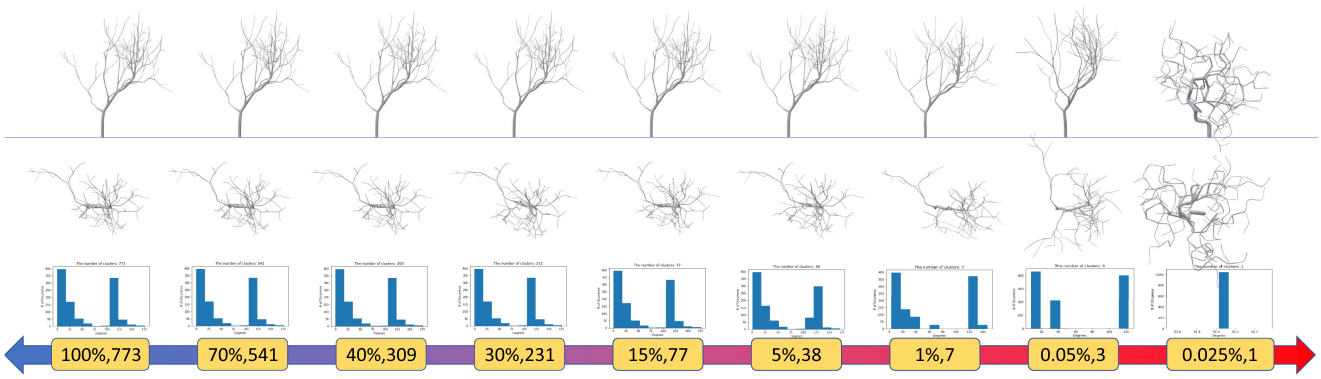

Fig. 5. Angle Quantization: The top two rows show the tree geometry of an Acacia Tree, and each step shows the change in the tree geometry with an increasing quantization level. The graph indicates the angle distribution and the value shows the percentage and the number of unique angles. The tree geometry does not visually change, even for extensive angle decimation.

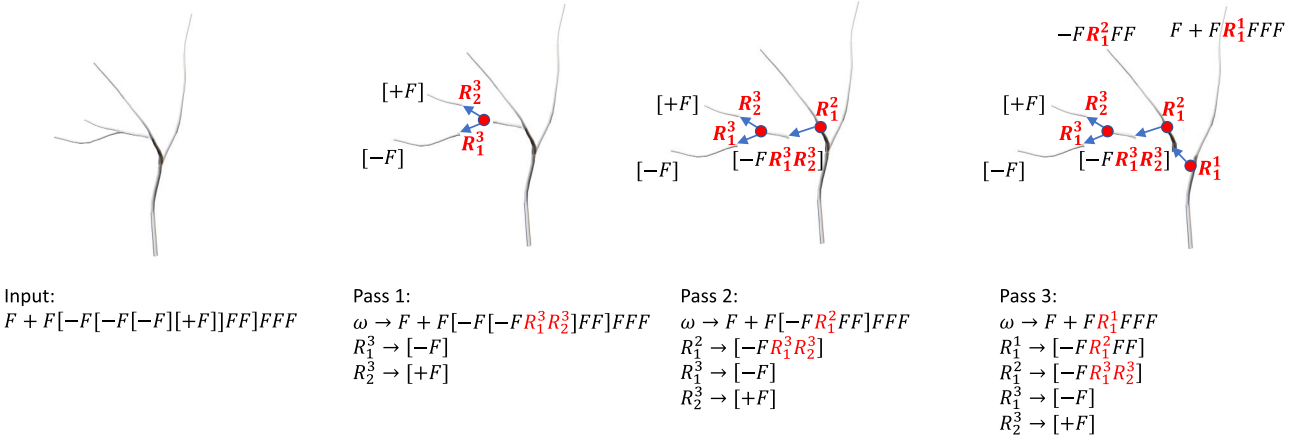

Fig. 6. A biological tree geometry and topology are encoded into a mathematical tree (left) and processed into bracket-free rules. In each pass, the inner-most parentheses are replaced by a non-terminal symbol R_i^g, where g is the recursion level (the Gravelius number) and i is the non-terminal symbol number.

the command needs to follow; e.g., $F(0.1)$ is interpreted as moving forward by one-tenth of the distance. The L-strings include a significant variance of the parameters that typically follow Gaussian distribution. We perform **quantization** of the parameters where each range of values is assigned a single representative value, and the number of values is given experimentally. We use k-means clustering of all parameters to find each value representative, and then we substitute each value with its corresponding k-means cluster center. An example in Figure 5 shows the effect of the quantization of the tree angles. The first two rows show the two views of the tree, and the histogram shows the angle distribution that indicates a bi-modal histogram with two prevailing values. It is further exemplified by reducing the number of parameters where the pair of the values on the lowest row shows the percentage of the parameters and the actual number of unique parameters. The quantization preserves the tree shape even for a relatively small number of clusters, such as reducing the number of unique values from 773 → 7, i.e., by 99%.

We then **de-parameterize** the quantized modules. We expand the L-system alphabet by replacing each module with a letter, i.e., $F(0.5)$ is substituted by $F05$. The total dataset size is reduced by compressing the numerical values from 31.2 GB to 13.8 GB (55.8%). The set of turtle commands from Table 1 is expanded accordingly, e.g., $F05$ is interpreted as "move forward by the distance 0.5". Note that each symbol then corresponds to a class of symbols: the meaning is the same, and it is given by the letter (Table 1), but the numbers correspond to the parameter. The de-parameterization is performed for all symbols from Table 1.

6.3 Rule Generation

We initially tested the **Long Short-Term Memory (LSTM)** [Hochreiter and Schmidhuber 1997]. We attempted to learn the string directly, but LSTM was (a) unable to learn the long string and (b) produced syntactically incorrect strings with missing brackets. The bracket pairs identify branches corresponding to a different hierarchy level, making geometric neighbors located far away in the string. In other words, while the letters A and G in the string $A[BCDEF]G$ are separated by seven letters, geometrically, AG are neighbors on the same branch, and $BCDEF$ is a branch attached at A. We tried the LSTM to learn the L-system rules, but it could not

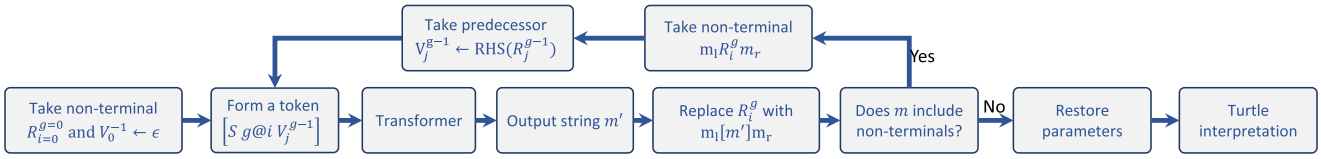

Fig. 7. L-string generation: The current string m is parsed, and all non-terminals are replaced by $[m']$ generated by the Transformer. The Transformer is invoked by forming a token indicating species S and the sequence identifying the non-terminal $g@i$ and its context V_j^{g-1}. When the string includes only terminal symbols, its parameters are restored, and the turtle interprets the string to produce the 3D geometry.

produce the hierarchies (see the Appendix for details and Table 8 for the LTSM model parameters). LSTM struggles to learn geometric information located at a great distance. In contrast, transformers are efficient in learning such distant geometric data because they use positional encoding. We also trained the Transformer directly on the L-strings, but it also did not generate the bracket pair correctly. We report the average loss values from three different experiments in Figure 21. Case 1 (loss 1.5) uses LSTM on L-system rules, Case 2 (loss 5.3) uses LSTM on L-strings, and Case 3 (loss 3.1) uses the Transformer on L-strings. While Case 1 has a smaller loss than the transformer, it cannot learn the rules.

Instead, we follow the topological organization of the L-string into brackets. We replace each level of hierarchy with a non-terminal symbol and an L-system rule representing a new level of hierarchy. It intuitively corresponds to removing the tree branches and organizing them into hierarchical rules, as depicted in Figure 6. The tree is parsed in g_{max} passes (levels of hierarchy), where $g_{max} = max\{g(v_i)\}\ v_i \in \mathcal{T}$ is the highest Gravelius order in the tree. In each pass, the deepest level of bracket pairs is replaced by non-terminal symbols R_g^i, where g is the Gravelius order level and i is the non-terminal symbol number. The right-hand-side of the rewriting rule (RHS) is the replaced string, and each string starts and ends with brackets as they encapsulate a branch. Note that the brackets are always located at a string's beginning and end. A string corresponding to a branch $[m]$ with the left context m_l and the right context m_r is substituted by R_i^g:

$$\begin{aligned} m_l[m]m_r &\Rightarrow m_l R_i^g m_r \\ id_g : R_i^g &\rightarrow [m]. \end{aligned} \tag{5}$$

This process is repeated until the L-string is bracket-free. An example in Figure 6 shows a 2D tree that has been encoded into a sequence of modules $F+F[-F[-F[-F][+F]]FF]FFF$ (parameters not shown). The $g_{max} = 3$ and in the first pass, the inner-most parenthesis $[-F]$ and $[+F]$ are replaced by rules $R_1^3 \rightarrow [-F]$, $R_2^3 \rightarrow [+F]$. The encoding ends when all parentheses have been removed.

The results of this step are the axiom of the grammar ω and rules encoding the branches on different levels of hierarchy. An important aspect of this encoding is that each branch includes information about the distribution of the subbranches: we count the distance between the non-terminal symbols (subbranches) on the string.

7 TRAINING

After all the input trees are encoded into L-system rules, we train a deep neural model that simulates the L-system string rewriting. It predicts the letters in the string corresponding to the tree geometry and learns the distribution of non-terminal symbols corresponding to the branches at the higher hierarchy levels.

Table 2. Training and Extracted Data: Species, Number of Trees, Maximum Recursion Depth (Gravelius Order g_{max}), Extracted L-string Length, Number of L-system Rules, and the Training Time

Species	# of Trees	g_{max}	String length avg, stdev	# of Rules avg, stdev	Train. time
Acacia	30 k	10	52 k, 16 k	314, 92	5 h 0 m
Birch	30 k	5	120 k, 12 k	834, 85	13 h 37 m
Maple	30 k	6	127 k, 17 k	860, 116	7 h 39 m
Oak	30 k	9	195 k, 62 k	1,225, 314	5 h 48 m
Pine	30 k	6	141 k, 20 k	1,041, 140	10 h 31 m
Walnut	5 k	13	108 k, 36 k	658, 193	15 h 55 m
Total	**155 k**		**475 k**	**4,529**	**42 h 30 m**

7.1 L-strings to Training Sequences

We convert each L-system rule R_i^g from Equation (5) into a *training sequence* Q_i^g

$$Q_i^g = \mathcal{S}_i^g \mathcal{V}_i^g \mathcal{E} \tag{6}$$

that consist of two *sub-sequences* and one *token*: the start $\mathcal{S}$, value $\mathcal{V}$, and the end $\mathcal{E} = [end]$. The model needs to connect two levels of hierarchy (branches), and this information needs to be included explicitly in the tokens. We put the rule's predecessor in the starting sequence and the RHS of the rule in the value sequence. Let's have the following rules:

$$R_0^{g-1} \rightarrow RHS(R_j^{g-1}) \tag{7}$$

$$R_i^g \rightarrow RHS(R_i^g), \tag{8}$$

where RHS indicates the right-hand-side of the rule. The $RHS(R_j^{g-1})$ is a sequence of strings of terminal ($m^*, m \in M$) and non-terminal symbols $m^* R_0^g\ m^* R_1^g \ldots m^*\ R_i^g$. The training sequence for each of the non-terminals is:

$$Q_i^g = \langle S\ g@i\ \rangle RHS(R_j^{g-1}) RHS(R_i^g)[end] \tag{9}$$

where symbols $S \in \{A, B, M, O, P, W\}$ are the species identifier (Acacia, Birch, Maple, Oak, Pine, and Walnut) (see Table 2), and the sequence $g@i$ in $\mathcal{S}$ is the rule ID. The rule R_0^0 corresponds to the tree trunk does not have a predecessor, and the training sequence is

$$Q_0^0 = \left\langle [S\ 0@0] RHS(R_0^0)[end] \right\rangle. \tag{10}$$

7.2 Model Architecture and Training

We use `seq2seq` [Sutskever et al. 2014] using the Transformer model, which follows [Vaswani et al. 2017], with Positional Encoding using sine and cosine functions and we show our model in Figure 22. Tokens are encoded based on their occurrence frequencies from the training data, and we use the Teacher Forcing method [Williams and Zipser 1989] to train the Transformer (see implementation in Section 9.1). We use early stopping to avoid overfitting or stop the training if a loss value does not improve for ten epochs.

During the training, we input a sequence Q_i^g (Equation (6)), and the model generates a new value sequence $\mathcal{V}'$. We measure the performance of the Transformer by calculating differences using cross entropy [Bishop and Nasrabadi 2006] between the ground truth sequence $\mathcal{V}_i^g$ and the generated sequence $\hat{\mathcal{V}}$, which are based on the same starting sequence $\mathcal{S}_i^g$.

Let $|P|$ be the number of unique parameters from the training data. The loss function is

$$loss = -\sum_{i=1}^{|P|} \mathcal{V}_i^g \log(\hat{\mathcal{V}}_i^g). \tag{11}$$

The loss function is designed to give equal weight to the generated sequence $\mathcal{V}'$ as varying weights produced a correct trunk but too short or long branches. This loss function helps the generation of well-balanced trees.

8 GENERATION

The generation process is similar to string rewriting in L-systems [Prusinkiewicz and Lindenmayer 1990] in that it takes the current string m and rewrites all non-terminal symbols in parallel with new strings. The main difference is that we do not parse a set of rewriting rules because the Transformer encodes the rules. Instead, the Transformer generates a sequence corresponding to the $RHS(R_i^g)$ of any rule R_i^g from the input data set per species (see Figure 7). We execute the Transformer by providing the species S, the non-terminal symbol R_i^g rule number, and its context (the predecessor R_j^{g-1}). It is important to note that the Transformer does not generate the exact copy of the RHS, but its sequence depends on the context. We use Temperature Sampling commonly used in text generation [Guo et al. 2018] that provides valid randomized tokens. In effect, the generated trees are also randomized. We use the temperature constant of 0.9 during the generation. Having the context V_j^{g-1}, the Transformer predicts the probability of the next tokens, and we apply the temperature sampling and then choose a token with the highest probability. Without the temperature sampling, the highest probability tokens from the model will be the same, causing the generation of identical trees. The tokens are generated until the Transformer outputs $\mathcal{E}$.

The key to deep L-string rewriting is how the Transformer tokens are formed. The current string m represents the tree geometry, and the deep generation starts by initializing m to the species S axiom $m \leftarrow R_0^0$. We parse the string, and for each non-terminal, we form the token $[S\ g@i\ V_j^{g-1}]$ corresponding to the rule R_i^g and its predecessor $V_j^{g-1} \leftarrow RHS(R_j^{g-1})$. Note that the axiom R_0^0 does not have a predecessor, so its predecessor is set to an empty string $V \leftarrow \epsilon$. The Transformer then responds with the output string m'.

The non-terminal symbol divides m into its left and right context $m_l R_i^g m_r$. We replace the newly generated string by encapsulating it into L-system brackets $[m']$, so the new string has the form $m \leftarrow m_l[m']m_r$. The newly generated sequence may add new non-terminal symbols corresponding to higher-level branching. The generation ends when m does not include non-terminal symbols.

We restore the parameters from the generated string by reversing the de-parameterization from Section 6.2. The turtle then interprets the string, and the tree geometry is generated. The generating process is guaranteed to terminate because the training data contains [*end*] at the end of every sequence. Figure 17 shows that the generated trees follow a similar distribution of the training data distribution.

9 IMPLEMENTATION AND RESULTS

9.1 Implementation

We use C++ to develop a simulator program to generate L-system string-based trees. The neural network is implemented in PyTorch version 1.12. We train the neural network on an Intel i9-12900 k at 4.8 GHz with Nvidia GeForce RTX 3090. All tree models were rendered by path tracing implemented in Nvidia OptiX 7.4.

Our implementation uses `seq2seq` [Sutskever et al. 2014] and the Transformer model based on Pytorch 1.12, which follows [Vaswani et al. 2017], and Positional Encoding using sine and cosine functions (see Figure 22 in the Appendix). Tokens are encoded based on their occurrence frequencies from the training data. We train the model with Adam optimizer [Kingma and Ba 2014] with a learning rate of 5×10^{-5}, and we stop the training if there are no improvements in loss values for ten epochs to prevent an overfitting. The model has three Transformer encoders and three decoder layers. We set four as the number of attention heads, 128 as the features for the encoder and decoder layers, 128 as the dimension of the feedforward network, and 0.1 for the dropout rate for the encoder and decoder layers (see Figure 22). We set 512 as the training batch size and used the Teacher Forcing [Williams and Zipser 1989] to train the Transformer.

9.2 Generated Trees

We used the Transformer to generate 200 trees for each species (1 k total) and show examples in Figures 11–16. We used the 200 trees for validation, as discussed below.

Training and Data Size: We train each Transformer model on $155k$ of tree models (Acacia, Birch, Maple, Oak, Pine) and $5k$ of tree models on Walnut (Table 2). We split the training data with an 80:20 ratio for training and validation. The training stops when overfitting occurs. We used $|S| = 6$ models, one for each tree species. Acacia, Birch, Maple, Oak, Pine, and Walnut tree models have been trained on 8, 5, 7, 5, 9, and 47 epochs (see Table 2). We maximize the 24 GB of the GPU memory by using the batch size of 512. The table also shows the amount of extracted data from the tree models: the string length storing the tree geometry and the number of the extracted bracket-free L-system rules.

Table 3. The Frequency of Occurrence of Each Terminal Symbol Parameter in the Input and the Generated Data Showed as Pairs μ, σ^2

Species	"+"	"-"	"^"	"&"	"/"	"\"	"F"
Acacia (input)	12.8,14.7	12.9,14.6	26.2,22.2	5.4,5.2	97.8,46.3	0.5,1.9	0.8,0.4
(generated)	12.8, 13.2	13.0,13.2	23.9,21.9	5.3,4.0	101.6,32.9	0.5,0.2	0.8,0.4
Birch (input)	1.0,0.8	1.1,0.9	23.0,22.0	1.1,0.6	116.2,44.6	74.2,88.2	0.8,0.4
(generated)	1.1,0.8	1.2,0.9	20.3,21.8	1.1,0.8	116.1,37.9	90.5,89.7	0.8,0.4
Maple (input)	2.0,1.8	2.2,2.0	23.4,23.9	2.5,1.9	116.6,40.4	75.7,88.2	0.8,0.4
(generated)	1.9,1.8	2.2,1.9	19.7,23.3	2.2,1.9	117.0,34.7	84.4,89.2	0.9,0.3
Oak (input)	16.5,7.3	1.3,1.9	11.0,14.2	2.1,2.0	117.7,33.3	78.6,88.3	0.8,0.4
(generated)	16.6,6.7	1.1,1.4	9.4,13.1	1.9,1.7	117.1,30.6	81.2,88.7	0.9,0.3
Pine (input)	1.8,1.9	1.8,1.9	41.0,38.9	2.5,2.0	116.0,43.4	73.2,87.3	0.8,0.3
(generated)	1.8,1.8	1.9,1.9	36.3,38.6	2.5,1.9	115.2,39.3	83.4,88.5	0.8,0.3
Walnut (input)	17.1,7.5	2.2,4.2	10.8,14.4	2.8,3.2	118.0,31.0	79.6,87.9	0.9,0.3
(generated)	17.3,7.3	1.1,3.2	12.0,14.6	3.2,3.3	118.8,30.6	86.1,87.9	0.9,0.3

9.3 Validation

A common way to compare strings is to pre-compute the discrete distribution of the following symbol given the current one and then use sampling to predict the next one. However, L-systems include hierarchies, and the brackets affect the string-edit distance, thus precluding us from using this approach. We validate our approach by comparing the similarity of distribution using extracted features of the training and generated data.

The L-strings encode 3D geometry, and we validate the ability of our model to reproduce the input tree geometric models (ground truth) by comparing their statistical distributions to the input data. The input data (Table 2) include 30 k trees per tree species and 5 k trees for Walnut. We generated 200 trees per species (Table 5) using the Transformer and compared them to the training data. In particular, we (a) measure their geometric properties (height, depth, geometry), (b) compare their perceived realism using ICTree perceptual metrics, and (c) show the three closest trees from the input dataset.

Tree Shape Global Features compare the height and width of the trees from the training set and the generated trees (Figure 19, Figure 36, Figure 37). The difference between the average width of the generated and real data is 19.7%. And the maximum difference is 36.8% for the Acacia tree species. The difference between the average height of the generated and real data is 21.9%, and the maximum difference is 37.1% for the Birch trees. The generated trees are about 2%–9% taller than the ground truth because the temperature sampling (Section 8) generates slightly more non-terminal symbols. This accumulates Forward (F) symbols in Figure 17. The discrepancies are within the range of the standard deviation. The differences from the tree shape global features between the ground truth trees and generated trees show our model does not generate the exact size of training data in terms of the tree shape.

Tree Depth refers to the complexity of tree branching, and it is measured as the maximum level of hierarchy from an L-system string (see Figure 19) and its density function of histogram in Figure 23. The difference between the average level of the hierarchy of the generated and real data is 6.3%. The maximum difference is 15.8% for the Oak tree species. Generating more branching levels can cause a thickening of the trunk and the lower branches.

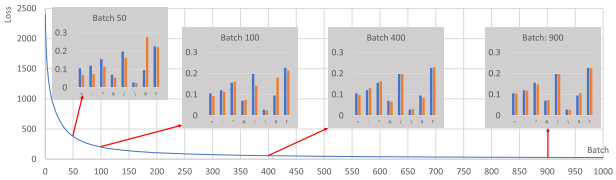

Fig. 8. Changes of the non-terminal symbol frequency during training of the input and the generated data.

Table 4. The Average Distance of Branchings

Species	Input avg, std	Generated avg, std	Δ [%]
Acacia	1.8, 1.2	1.8, 1.3	0
Birch	1.1, 0.4	1.1, 0.4	0
Maple	1.2, 0.6	1.2, 0.5	0
Oak	1.6, 1.1	1.5, 1.1	6
Pine	1.0, 0.4	1.1, 0.4	10
Walnut	1.9, 1.3	1.6, 1.0	15

Terminal Symbols encode the tree geometry and we compared the frequency of each terminal symbol in the input data of 30 k trees for each species and the 200 trees generated by our method. Figure 17 and Table 3 show the comparison of their distributions. The generated trees have similar averages and standard deviations of the terminal symbol distributions. In particular, the average difference of angles is Acacia: 3.7%, Birch 7.6%, Maple 7.3%, Pine 5.8%, Oak 6.8%, Walnut 5.4%, and the overall average difference is 6.1%. The average difference in lengths of branches is Acacia: 1.7%, Birch 2.1%, Maple 4.2%, Pine 3.1%, Oak 2.8%, and Walnut 1.0% and the overall average difference is 2.5%. We also report the distribution of values of terminal symbols. Figures 24–29 in the Supplementary shows the density function of the distribution of the terminal symbols. The overall average difference in angles and lengths of branches shows that our model generates trees with a similar distribution of geometric attributes.

Figure 8 shows how the frequency of the terminal symbols changes during the training process. The initial distribution of the

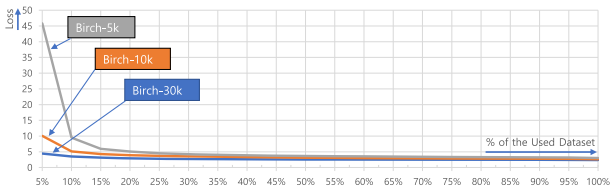

Fig. 9. The effect of the number of input data trees (5 k-10 k-30 k) on the validation loss. The x-axis shows the percentage of dataset size used for the training. The larger training dataset shows better performance in general.

training data (red) varies significantly from the input (blue). However, after about 400 iterations, the training loss shows that the model is converging, and after 900 iterations, the differences between the distribution of training data and the generated data are negligible.

Non-terminal Symbols define tree branching, and a qualitative indicator of their distribution is their average distance in each tree. We hypothesize that the distribution of the branchings in the input and the generated data should be similar. We measure the distances by traversing the trees from the root and calculating the geodesic distance of the next branch. Note that a simple count of the non-terminals would not report a correct result, as the two strings $F(2)$ and $F(1)F(1)$ have the same geometric meaning. We perform this task recursively for every tree and report the average and the standard deviation in Figure 18 and Table 4 we also report its density function of histogram in Figure 20. The overall average distance difference is 5.2% that indicates an agreement between the input and the generated data. The distributions of the non-terminal symbols correspond to the branching elements, and preserving their distributions shows that the branching is also preserved (see Table 4).

Figure 9 shows the effect of the amount of training data on the **validation loss**. We trained the model using 5 k, 10 k, and 30 k Birch trees. Each model takes 38 min, 53 min, and 150 min, respectively. After the model trained on 100% of available trees in each dataset, their best loss values were 3.179, 2.751, and 2.437 for 5,000, 10, 000 and 30,000 trees. The loss values of the 5 k trees model and the 10 k trees model are 30.4% and 12.9% higher than the 30 k trees model. We experiment with a weighted loss using angle and distance tokens. Interestingly, the weight parameter choice does not significantly affect the validation loss. We experimented with the following combinations: [angle, distance] [0.1,0.9], [0.5,0.5], and [0.9, 0.1] leading to validation losses of 1.013, 1.012, and 1.013.

Depth Level Distribution compares the Gravelius order of each tree (see all figures in the Supplementary Materials). The figures show the models learned well by comparing averaged values between the parameters because of their small differences. Moreover, generated trees have a larger standard deviation, indicating that the models did not memorize the ground truth data. Still, they can generate new trees likely in the training data but not the same trees.

Tree Similarity shows a side-by-side comparison of a randomly selected generated tree with three samples from the training dataset that were geometrically closest. Tree geometric similarity is an open research problem, and we adopted the tree geometry

Table 5. Transformer-generated Trees: Species, Number of Trees, Maximum Recursion Depth (Gravelius Order g_{max}), and the Training Time

Species	Number of Trees	g_{max} g_{max}	Gen. time all	Gen. time single tree
Acacia	200	10	37.80 min	11.34 s
Birch	200	5	55.60 min	16.68 s
Maple	200	6	53.73 min	16.12 s
Oak	200	9	58.27 min	17.48 s
Pine	200	6	68.47 min	20.54 s
Walnut	200	13	62.13 min	18.64 s
total	**1,200**		**56.00 min**	**16.80 s**

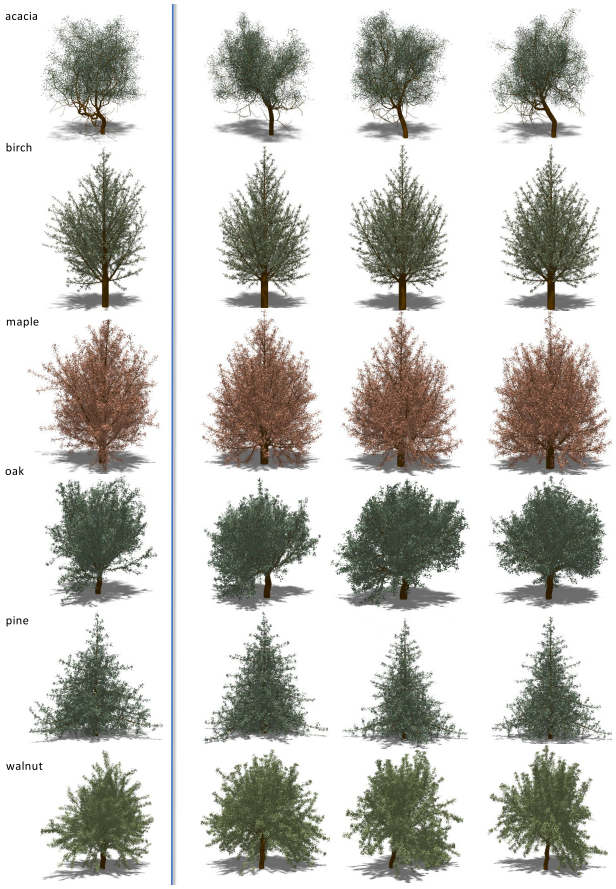

Fig. 10. Visual comparison of trees generated by latent L-systems (left) and three geometrically closest trees from the validation dataset (right).

decomposition into a list of features introduced in Polasek et al. [2021]. In particular, the tree is divided into a list of chains that are sequences of segments defining a branch. The feature vector stores each chain's length, curvature, maximum and minimum angle, straightness, and the number of segments. We also store the global feature, that is, the maximum depth of segments (Gravelius number) of the tree. We refer authors to Polasek et al. [2021, Section 4.3] for more details on how the features are calculated. The

Table 6. Perceived Tree Realism Metrics ICTree for the Input Data and the Generated Models

Species	Input (avg, stdev)	Generated (avg, stdev)
Acacia	0.96, 0.01	0.97, 0.01
Birch	0.94, 0.01	0.98, 0.01
Maple	0.94, 0.02	0.99, 0.01
Oak	0.92, 0.03	0.96, 0.02
Pine	0.94, 0.01	0.98, 0.01
Walnut	0.95, 0.02	0.94, 0.02
Total	**0.94, 0.02**	**0.97, 0.01**

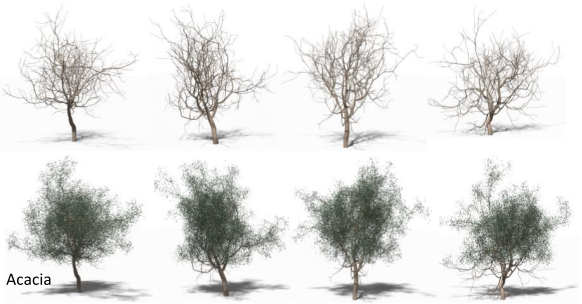

Fig. 11. Acacia tree models generated with our system.

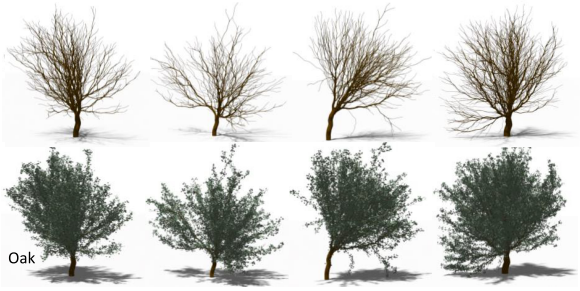

Fig. 12. Oak tree models generated with our system.

feature vector of the selected generated tree is used to find the three nearest neighbors in the training data set by using L^2 metrics. Results in Figure 10 show the Transformer-generated 3D geometry (left) and the three nearest neighbors from the validation dataset unseen during the training.

ICTree is a recent work [Polasek et al. 2021] that outputs perceived visual realism of tree models from their geometry and images. The deep learning-based method provides a value of perceived tree realism. We hypothesize that the ICTree values of the input dataset should not vary from the generated models. The input data for acacia, birch, tree, maple, pine, and walnut have 0.96, 0.94, 0.94, 0.92, 0.95, and 0.95. The average score for the generated trees is 0.94.(0-non-realistic, 1-realistic) from its geometry.

Table 6 shows the perceived visual realism values per species for the input and the generated models. Our results show that the average perceived realism of all ground truth trees is 0.94, and the generated ones are 0.97 with a difference of 3.20%, with the

Fig. 13. Maple tree models generated with our system.

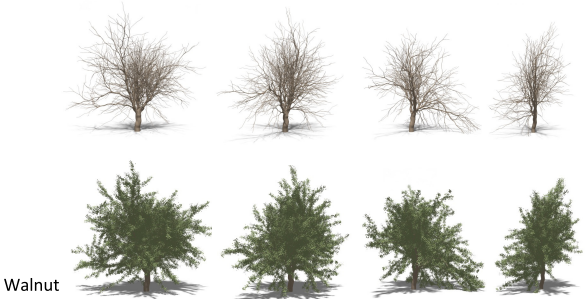

Fig. 14. Walnut tree models generated with our system.

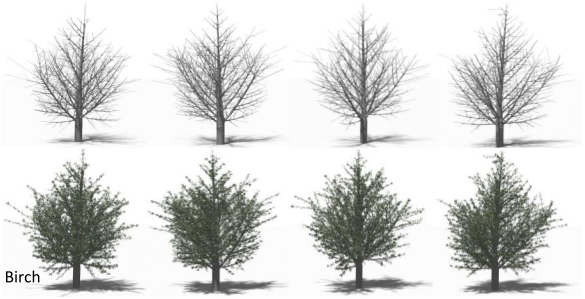

Fig. 15. Birch tree models generated with our system.

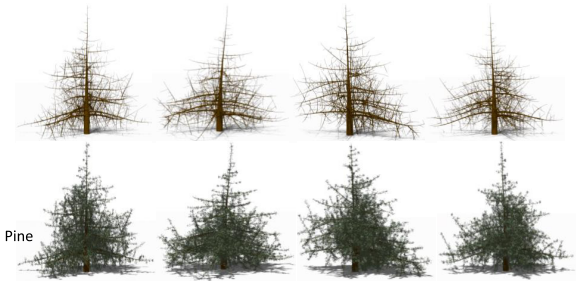

Fig. 16. Pine tree models generated with our system.

highest difference for Maple of 5.31%. The standard deviation of the ground truth is 0.02, and the generated trees are 0.01. While the difference seems large, it is small compared to the average value. ICTree metrics for the ground truth data are slightly smaller

Table 7. Tree Features from ICTree [Polasek et al. 2021], the Input and the Generated Data Showed as Pairs μ, σ^2

Species	Internode Count	Length	Max Child Angle Total	Max Child Angle Total	Max Sibling Angle Total	Total Length	Slope	Straightness
Acacia (input)	1.547,0.093	1.911,0.088	0.343,0.024	0.775,0.025	0.343,0.024	1.790,0.075	1.056,0.058	0.924,0.003
(generated)	1.328,0.119	2.320,0.147	0.456,0.032	0.786,0.034	0.457,0.032	2.139,0.129	1.095,0.089	0.919,0.005
Birch (input)	0.783,0.029	1.614,0.062	0.257,0.021	0.473,0.012	0.258,0.020	1.574,0.059	0.884,0.027	0.954,0.001
(generated)	0.673,0.029	1.932,0.046	0.336,0.014	0.519,0.011	0.344,0.014	1.873,0.044	0.861,0.076	0.953,0.001
Maple (input)	0.814,0.049	1.861,0.101	0.334,0.032	0.553,0.020	0.338,0.031	1.800,0.095	0.986,0.026	0.949,0.002
(generated)	0.613,0.039	2.392,0.089	0.472,0.021	0.625,0.015	0.476,0.019	2.297,0.086	0.905,0.096	0.946,0.001
Oak (input)	1.126,0.176	2.388,0.326	0.339,0.042	0.518,0.026	0.342,0.042	2.314,0.309	1.029,0.112	0.964,0.002
(generated)	0.836,0.089	3.034,0.314	0.412,0.021	0.533,0.014	0.412,0.021	2.934,0.304	0.881,0.122	0.963,0.002
Pine (input)	0.799,0.027	1.631,0.061	0.418,0.034	0.826,0.021	0.427,0.033	1.512,0.052	1.525,0.027	0.876,0.003
(generated)	0.645,0.021	1.925,0.046	0.633,0.029	0.943,0.017	0.639,0.028	1.731,0.040	1.542,0.036	0.853,0.004
Walnut (input)	1.247,0.122	2.637,0.197	0.365,0.027	0.560,0.022	0.371,0.028	2.540,0.183	1.130,0.124	0.960,0.002
(generated)	1.124,0.078	2.401,0.098	0.352,0.019	0.555,0.017	0.357,0.020	2.313,0.092	1.120,0.095	0.960,0.002

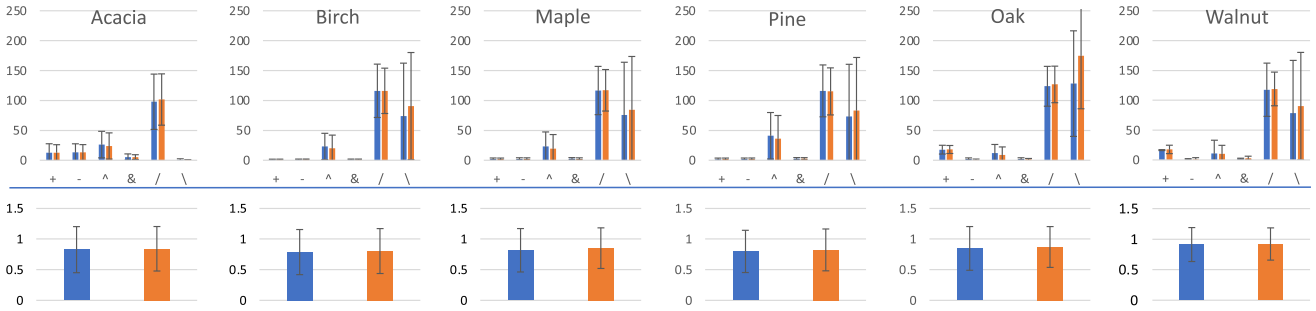

Fig. 17. Comparison of the distributions of the terminal symbols in the input and generated data. The top row shows the distribution of the angles and the bottom row shows the distribution of the "F" symbol for the input and the generated data.

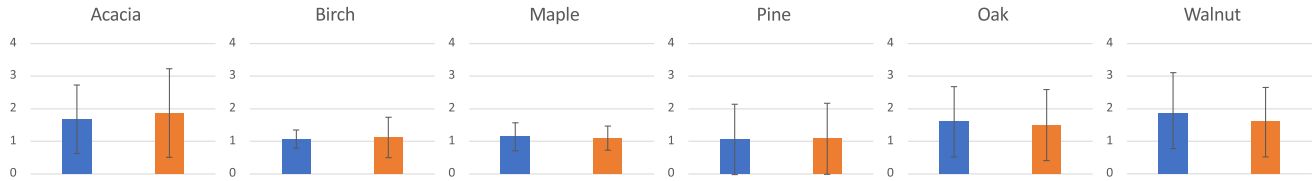

Fig. 18. Comparison of the distributions of the non-terminal symbols (branchings) in the input and the generated data.

than our generated trees. The similarity of the scores and their distributions shows that the models would be perceived as realistic and the small standard deviation (0.01) indicates a low number of outliers.

Tree Features. We also calculate the comprehensive set of extracted geometric features by using the approach from Polasek et al. [2021], Stava et al. [2014]. In particular, we compare the chain internode count, chain length, chain max child angle total, chain max sibling angle total, chain total length, chain slope, and chain straightness (see Table 7). The overall average and standard deviation from the ground truth with tree features is 1.04 and 0.65, respectively. Also, the model-generated tree features agree with the ground truth at 92.3% with 1.12 and 0.75 as its overall average and standard deviation. The max child angle total has the largest average percentage difference of 30.25%, and straightness has the smallest average percentage difference, which is 0.61%.

10 CONCLUSIONS AND FUTURE WORK

L-systems are a powerful mathematical formalism, with the main drawback being the unintuitiveness of the tree model description. Our approach uses L-systems as an intermediate representation that is invisible to the user as it replaces the L-system rules with a deep neural model that can be trained from real data. We train our model on 155 k 3D tree geometries. We show that it can generate an arbitrary number of tree geometries similar to the training data as validated by comparing the topology, geometry, and perceived realism. Our key novelty is in representing the tree geometry as a string that can be used by a common deep neural model. We

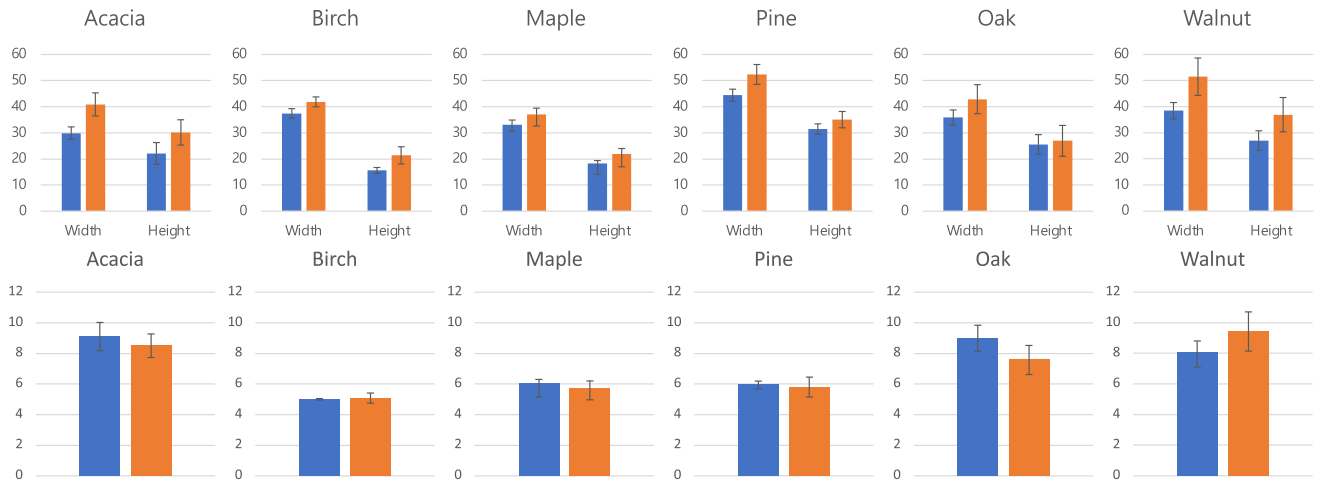

Fig. 19. Comparison of the tree height and width (upper row) and the recursion depth (lower row) in the input and generated data and it shows the input and the generated data. The bar shows the average of values and the line shows the standard deviation of values.

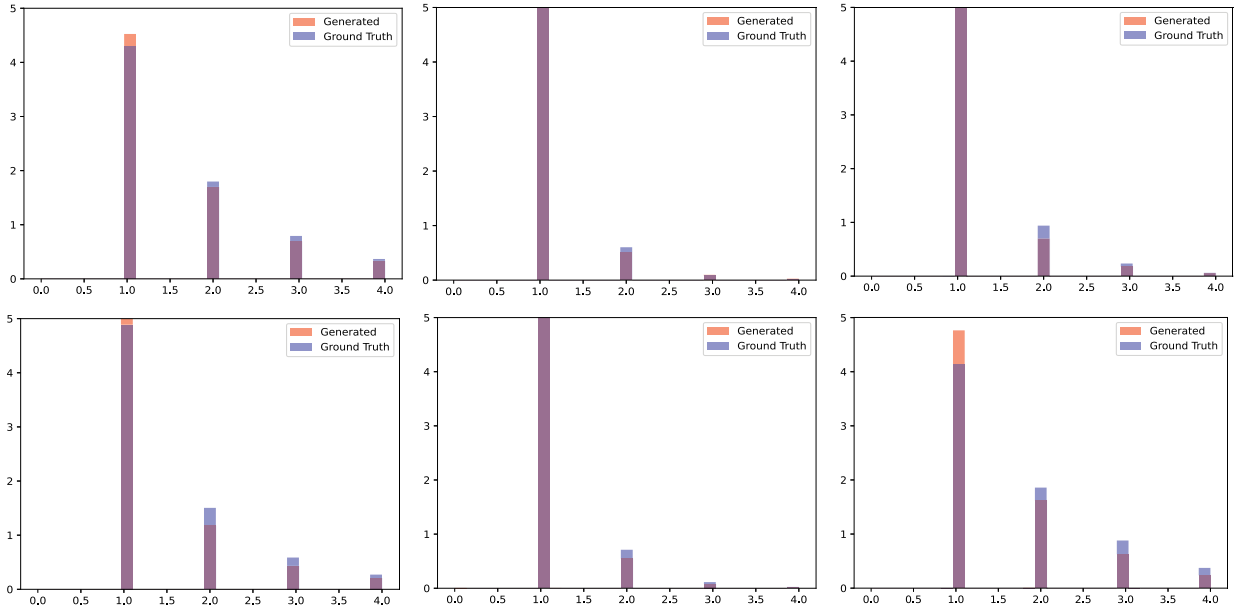

Fig. 20. A histogram of non-terminal symbols for (Top)Acacia, Birch, Maple and (Bottom)Oak, Pine and Walnut trees.

show that the generated trees have similar geometric and topological properties and similar perceptual measures as the training data.

Our model has several limitations. One of them is the common limitation of procedural models, i.e., low control. The user sends an input token to the Transformer, and the deep neural model responds with a tree geometry. The second limitation is the input training set. While computer graphics-based 3D tree models are becoming realistic, it would be helpful to train our deep model on a set of real tree geometries. However, to the best of our knowledge, such a dataset does not exist. Another limitation is the relatively slow generation time of our models (seconds). While existing procedural methods written in C++ and CUDA are capable of generating 3D tree models interacting with wind [Pirk et al. 2014], our Python-based implementation is much slower.

There are several possible avenues for future work. The first is in providing control over the space of the generated trees. While procedural models have shown flexibility in interactive model generation [Longay et al. 2012], our deep model provides the 3D model geometry at once. Deep neural models are often challenging to be employed interactively, and a potential future work should address interactive editing with a deep neural model. It would also be interesting to employ our work in different tasks, such as tree reconstruction. Another possible future work should

address the successive tree hierarchy generation and the environmental factors. We have shown that the Transformer can encode string hierarchies that represent topology and 3D geometry. An exciting area of future work would show how different hierarchies or application areas could be coupled with deep neural models.

A APPENDIX

LSTM: We designed an LSTM models with various parameters combinations with `seq2seq`. We report sets of hyper-parameter in Table 8, where each column represents the number of layers in the Encoder and Decoder, Encoder embedding dimension, Decoder embedding dimension, and the hidden layer dimension. Our first generated sequence, which corresponds to the tree trunk, was impossible to generate by the trained LSTM-based model because it does not contain enough information to deal with unbalanced parenthesis. The generated sequence was impossible to convert back into the trunk, which is needed to generate the rest of the tree. Moreover, we fed the correct data to the trained model, but it was not capable of recreating the expected output.

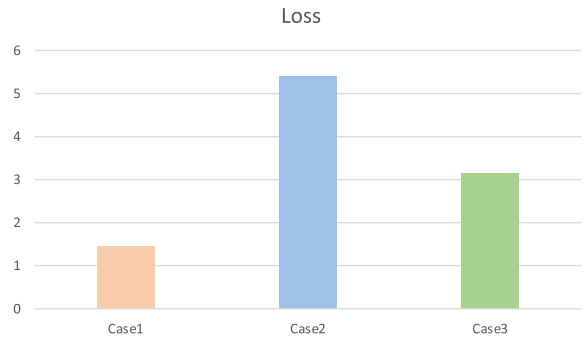

Fig. 21. Performance Comparison on Different Model Architecture and Data Representations. Case 1 uses LSTM on L-system rules, Case 2 uses LSTM on L-strings, and Case 3 uses the Transformer on L-strings.

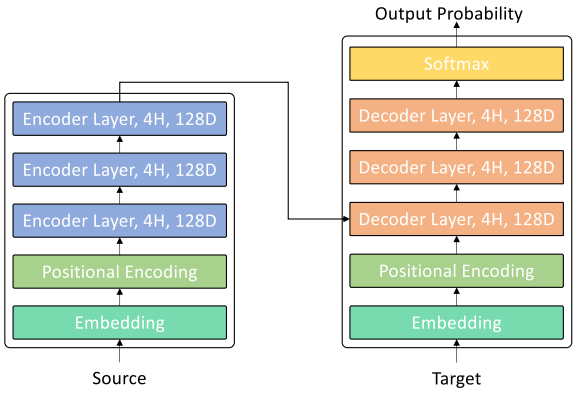

Fig. 22. Model Architecture (H: the number of attention heads, D: the number of dimensions).

Table 8. Hyper-parameter of LSTM Models

Layers	Enc. Embedding	Dec. Embedding	Dimension
6	512	512	256
6	128	128	64
6	128	128	256
6	32	32	256
3	256	256	512
3	64	64	128
3	64	64	64
2	32	32	256
1	256	256	512
1	128	128	256
1	64	64	128
1	64	64	64

ACKNOWLEDGMENTS

We want to thank Songlin Fei for his insight and help at the initial stages of this project.

REFERENCES

M. Aono and T. L. Kunii. 1984. Botanical tree image generation. *IEEE Computer Graphics and Applications* 4, 5 (1984), 10–34.

J. Arvo and D. Kirk. 1988. Modeling plants with environment-sensitive automata. In *Proceedings of the Ausgraph*. 27–33.

Jason Bernard and Ian McQuillan. 2021. Techniques for inferring context-free lindenmayer systems with genetic algorithm. *Swarm and Evolutionary Computation* 64 (2021), 100893.

Christopher M. Bishop and Nasser M. Nasrabadi. 2006. *Pattern Recognition and Machine Learning*. Vol. 4. Springer.

Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. 2010. A connection between partial symmetry and inverse procedural modeling. In *Proceedings of the ACM SIGGRAPH 2010 Papers*. 1–10.

Nadav Brandes, Dan Ofer, Yam Peleg, Nadav Rappoport, and Michal Linial. 2022. ProteinBERT: A universal deep-learning model of protein sequence and function. *Bioinformatics* 38, 8 (2022), 2102–2110. DOI: https://doi.org/10.1093/bioinformatics/btac020 arXiv:https://academic.oup.com/bioinformatics/article-pdf/38/8/2102/49009610/btac020.pdf

Jiacheng Chen, Chen Liu, Jiaye Wu, and Yasutaka Furukawa. 2019. Floor-sp: Inverse cad for floorplans by sequential room-wise shortest path. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2661–2670.

Phillippe De Reffye, Claude Edelin, Jean Françon, Marc Jaeger, and Claude Puech. 1988. Plant models faithful to botanical structure and development. *ACM Siggraph Computer Graphics* 22, 4 (1988), 151–158.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv: 1810.04805. Retrieved from https://arxiv.org/abs/1810.04805

Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. 2018. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Shenglan Du, Roderik Lindenbergh, Hugo Ledoux, Jantien Stoter, and Liangliang Nan. 2019. AdTree: Accurate, detailed, and automatic modelling of laser-scanned trees. *Remote Sensing* 11, 18 (2019), 2074.

Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. 2018. InverseCSG: Automatic conversion of 3D models to CSG trees. *ACM Transactions on Graphics* 37, 6 (2018), 16 pages. DOI: https://doi.org/10.1145/3272127.3275006

William Fedus, Ian Goodfellow, and Andrew M. Dai. 2018. Maskgan: Better text generation via filling in the_. arXiv:1801.07736. Retrieved from https://arxiv.org/abs/1801.07736

Benjamin G. Fitch, Patrick Parslow, and Karsten Ø. Lundqvist. 2018. Evolving complete L-systems: Using genetic algorithms for the generation of realistic plants. In *Proceedings of the Artificial Life and Intelligent Agents Symposium*. Springer, 16–23.

Lin Gao, Jie Yang, Tong Wu, Yu-Jie Yuan, Hongbo Fu, Yu-Kun Lai, and Hao Zhang. 2019. SDM-NET: Deep generative network for structured deformable mesh. *ACM Transactions on Graphics* 38, 6 (2019), 1–15.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in Neural Information Processing Systems* 27 (2014), 2672–2680.

H. Gravelius. 1914. Grundrib der gasamten Gewässerkunde,band I: Flubkunde. *kompendium of Hydrology* 1 (1914), 265–278.

Ned Greene. 1989. Voxel space automata: Modeling with stochastic growth processes in voxel space. *Proceedings of the 16th annual Csonference on Computer Graphics and Interactive Techniques* 23, 3 (1989), 175–184.

Jianwei Guo, Haiyong Jiang, Bedrich Benes, Oliver Deussen, Xiaopeng Zhang, Dani Lischinski, and Hui Huang. 2020. Inverse procedural modeling of branching structures by inferring l-systems. *ACM Transactions on Graphics* 39, 5 (2020), 13 pages.

Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Long text generation via adversarial training with leaked information. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Ralf Habel, Alexander Kusternig, and Michael Wimmer. 2009. Physically guided animation of trees. In *Proceedings of the Computer Graphics Forum*. Wiley Online Library, 523–532.

Torsten Hädrich, Daniel T. Banuti, Wojtek Pałubicki, Sören Pirk, and Dominik L. Michels. 2021. Fire in paradise: Mesoscale simulation of wildfires. *ACM Transactions on Graphics* 40, 4 (2021), 15 pages. DOI: https://doi.org/10.1145/3450626.3459954

Torsten Hädrich, Bedrich Benes, Oliver Deussen, and Sören Pirk. 2017. Interactive modeling and authoring of climbing plants. 36, 2 (2017), 49–61. DOI: https://doi.org/10.1111/cgf.13106 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13106

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

Matthew Holton. 1994. Strands, gravity and botanical tree imagery. In *Proceedings of the Computer Graphics Forum*. Wiley Online Library, 57–67.

R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. 2020. ShapeAssembly: Learning to generate programs for 3D shape structure synthesis. *ACM Transactions on Graphics* 39, 6 (2020), 20 pages. DOI: https://doi.org/10.1145/3414685.3417812

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (2021), 583–589.

Radoslaw Karwowski and Przemyslaw Prusinkiewicz. 2004. The l-system-based plant-modeling environment l-studio 4.0. In *Proceedings of the 4th International Workshop on Functional-structural Plant Models*. UMR AMAP Montpellier, France, 403–405.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv:1412.6980. Retrieved from https://arxiv.org/abs/1412.6980

Bosheng Li, Jacek Kałużny, Jonathan Klein, Dominik L. Michels, Wojtek Pałubicki, Bedrich Benes, and Sören Pirk. 2021. Learning to reconstruct botanical trees from single images. *ACM Trans. Graph.* 40, 6, Article 231 (December 2021), 15 pages. https://doi.org/10.1145/3478513.3480525

Bosheng Li, Jonathan Klein, Dominik L. Michels, Bedrich Benes, Sören Pirk, and Wojtek Pałubicki. 2023. Rhizomorph: The coordinated function of shoots and roots. *ACM Trans. Graph.* 42, 4, Article 59 (August 2023), 16 pages. https://doi.org/10.1145/3592145

Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. 2017. Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics* 36, 4 (2017), 1–14.

Yang Li, Quan Pan, Suhang Wang, Tao Yang, and Erik Cambria. 2018. A generative model for category text generation. *Information Sciences* 450 (2018), 301–315.

Aristid Lindenmayer. 1968. Mathematical models for cellular interaction in development. *Journal of Theoretical Biology* Parts I and II, 18, 3 (1968), 280–315.

Steven Longay, Adam Runions, Frédéric Boudon, and Przemyslaw Prusinkiewicz. 2012. TreeSketch: Interactive procedural modeling of trees on a tablet. In *Proceedings of the SBIM Expressive*. 107–120.

Jean-Eudes Marvie, Julien Perret, and Kadi Bouatouch. 2005. The FL-system: A functional l-system for procedural geometric modeling. *The Visual Computer* 21, 5 (2005), 329–339.

Ryoko Minamino and Masaki Tateno. 2014. Tree branching: Leonardo da vinci's rule versus biomechanical models. *PloS One* 9, 4 (2014), e93535.

Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy J. Mitra, and Leonidas J. Guibas. 2019. StructureNet: hierarchical graph networks for 3D shape generation. *ACM Trans. Graph.* 38, 6, Article 242 (December 2019), 19 pages. https://doi.org/10.1145/3355089.3356527

Radomír Měch and Przemyslaw Prusinkiewicz. 1996. Visual models of plants interacting with their environment. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. ACM, New York, NY, 397–410. DOI: https://doi.org/10.1145/237170.237279

Charlie Nash, Yaroslav Ganin, S.M. Ali Eslami, and Peter Battaglia. 2020. Polygen: An autoregressive generative model of 3d meshes. In *Proceedings of the International Conference on Machine Learning*. PMLR, 7220–7229.

Gabriela Ochoa. 1998. On genetic algorithms and lindenmayer systems. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*. Springer, 335–344.

Makoto Okabe, Shigeru Owada, and Takeo Igarashi. 2007. Interactive design of botanical trees using freehand sketches and example-based editing. In *Proceedings of the ACM SIGGRAPH 2007 Courses*. Association for Computing Machinery, New York, NY, 26–es. DOI: https://doi.org/10.1145/1281500.1281537

Peter E. Oppenheimer. 1986. Real time design and animation of fractal plants and trees. *ACM SiGGRAPH Computer Graphics* 20, 4 (1986), 55–64. DOI: https://doi.org/10.1145/15886.15892

Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Brendan Lane, Radomír Měch, and Przemyslaw Prusinkiewicz. 2009. Self-organizing tree models for image synthesis. *ACM TOG* 28, 3 (2009), 10 pages.

Sören Pirk, Bedrich Benes, Takashi Ijiri, Yangyan Li, Oliver Deussen, Baoquan Chen, and Radomir Měch. 2016. Modeling plant life in computer graphics. In *Proceedings of the ACM SIGGRAPH 2016 Courses*. 180 pages. DOI: https://doi.org/10.1145/2897826.2927332

Sören Pirk, Till Niese, Torsten Hädrich, Bedrich Benes, and Oliver Deussen. 2014. Windy trees: Computing stress response for developmental tree models. *ACM Transactions on Graphics* 33, 6 (2014), 11 pages. DOI: https://doi.org/10.1145/2661229.2661252

Sören Pirk, Ondrej Stava, Julian Kratt, Michel Abdul Massih Said, Boris Neubert, Radomír Měch, Bedrich Benes, and Oliver Deussen. 2012. Plastic trees: Interactive self-adapting botanical tree models. *ACM TOG* 31, 4 (2012).

Tomas Polasek, David Hrusa, Bedrich Benes, and Martin Cadik. 2021. ICTree: Automatic perceptual metrics for tree models. *ACM Transaction on Graphics* 40, 6 (2021), 15 pages. DOI: https://doi.org/10.1145/3478513.3480519

Przemyslaw Prusinkiewicz. 1986. Graphical applications of l-systems. In *Proceedings on Graphics Interface '86/Vision Interface '86*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 247–253. Retrieved from http://dl.acm.org/citation.cfm?id=16564.16608

Przemyslaw Prusinkiewicz, Mark S. Hammel, and Eric Mjolsness. 1993. Animation of plant development. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. 351–360.

Przemyslaw Prusinkiewicz and Jim Hanan. 1990. Visualization of botanical structures and processes using parametric l-systems. In *Proceedings of the Scientific Visualization and Graphics Simulation*. 183–201.

Przemyslaw Prusinkiewicz and Aristid Lindenmayer. 1990. *The Algorithmic Beauty of Plants*. Springer–Verlag, New York. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. de Boer, and L. Mercer.

Ruben M. Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. 2014. A survey on procedural modelling for virtual worlds. *Computer Graphics Forum* 33, 6 (2014), 31–50. DOI: https://doi.org/10.1111/cgf.12276 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12276

Alvy Ray Smith. 1984. Plants, fractals, and formal languages. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, New York, NY, 1–10. DOI: https://doi.org/10.1145/800031.808571

Ondrej Stava, Sören Pirk, Julian Kratt, Baoquan Chen, Radomír Měch, Oliver Deussen, and Bedrich Benes. 2014. Inverse procedural modelling of trees. In *Proceedings of the Computer Graphics Forum*. Wiley Online Library, 118–131.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*. MIT Press, Cambridge, MA, 3104–3112.

Carlos A. Vanegas, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Paul Waddell. 2012. Inverse design of urban procedural models. *ACM Transactions on Graphics* 31, 6 (2012), 1–11.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates Inc., Red Hook, NY, 6000–6010.

Guan Wang, Hamid Laga, Jinyuan Jia, Ning Xie, and Hedi Tabia. 2018. Statistical modeling of the 3D geometry and topology of botanical trees. *Computer Graphics Forum* 37, 5 (2018), 185–198.

Jason Weber and Joseph Penn. 1995. Creation and rendering of realistic trees. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. Association for Computing Machinery, New York, NY, 119–128. DOI: https://doi.org/10.1145/218380.218427

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* 1, 2 (1989), 270–280. DOI: https://doi.org/10.1162/neco.1989.1.2.270

Fuzhang Wu, Dong-Ming Yan, Weiming Dong, Xiaopeng Zhang, and Peter Wonka. 2014. Inverse procedural modeling of facade layouts. *ACM Trans. Graph.* 33, 4, Article 121 (July 2014), 10 pages. https://doi.org/10.1145/2601097.2601162

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation?. In *Proceedings of the Advances in Neural Information Processing Systems*. M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang,

and J. Wortman Vaughan (Eds.), Vol. 34, Curran Associates, Inc., 28877–28888. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2021/file/f1c1592588411002af340cbaedd6fc33-Paper.pdf

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. 2019. Graph transformer networks. *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, Article 1073, 11983–11993.

Xiaochen Zhou, Bosheng Li, Bedrich Benes, Songlin Fei, and Soeren Pirk. 2023. DeepTree: Modeling trees with situated latents. *IEEE Transactions on Visualization and Computer Graphics* 01 (2023), 1–14. https://doi.org/10.1109/TVCG.2023.3307887

Received 24 October 2022; revised 5 August 2023; accepted 29 September 2023