Interactive Arrangement of Botanical L-System Models

Joanna L. Power *A.J. Bernheim Brush* *Przemyslaw Prusinkiewicz*[†] *David H. Salesin*

University of Washington [†]University of Calgary

Abstract

In this paper, we explore the problem of interactively manipulating plant models without sacrificing their botanical accuracy. The primary technical contribution of the paper is a method for interactively manipulating plant structures using a inverse-kinematics optimization technique. The branches of the plant are endowed with flexural and torsional stiffnesses, and these are used in the IK optimization. We demonstrate our approach with several examples of plant models arranged in this fashion.

Keywords: botanical modeling, L-system, plant arrangement, inverse-kinematics, optimization, interactive techniques

1 Introduction

Lindenmayer-systems, or L-systems for short, were introduced as a theoretical model of plant development [5]. In the hands of computer graphics researchers, L-systems have evolved into a powerful tool for creating biologically faithful and visually realistic models of plants, capable of simulating their growth and interaction with the environment [7, 10, 11].

The power of L-systems lies in their ability to generate complicated structures from a small number of rules, but the cost of this brevity is the lack of precise control over the final form. This is a drawback of L-system models in applications such as illustration, scene design, and animation — areas in which a designer's aesthetic vision is the priority. In such areas, what is needed is a plant model that can grow and respond to its environment, yet that can be easily adjusted and controlled. Our application *ilsa* (interactive L-string arranger) allows a user to interactively manipulate a plant model while preserving its botanical accuracy and behavior, improving the usefulness of L-systems in domains beyond biological modeling.

The ability to interact with L-system models extends their utility within the domain of botanical modeling as well. A model can be grown for a certain number of generations, arranged or pruned, and then grown some more. The final form of such a model expresses both the developmental behavior of the plant and the effects of human intervention.

Our project ties together threads of related work from two areas of computer graphics: plant modeling using L-systems and realistic model manipulation. The majority of work on L-system modeling is that of Prusinkiewicz *et al.* [7, 10, 11]. In the broad area of interactive manipulation, the most closely related work is that of Zhao and Badler, who developed a system for the interactive manipulation of jointed figures using inverse-kinematics, and that of Harada *et al.*, who invented a technique for interactive manipulation of grammar-based models [15, 3].

The remainder of the paper is organized as follows: Section 2 provides relevant background information on L-systems. Section 3 describes *ilsa* and the arrangement process. Section 4 presents some results and discusses their creation. Section 5 concludes with a discussion of contributions and possible directions for future work.

2 L-systems and L-strings

An L-system consists of a set of textual rules called *productions* that describe the development of plant branches, leaves, flowers, and other components. In a *generation phase* these productions are applied in a sequence of *derivation steps* to the initial string, called the *axiom*. The state of the L-system model after any number of steps is encoded in a string of symbols, called the *L-string*. In a subsequent *interpretation phase* the L-string is converted to a geometric representation of a plant.

L-strings encode form using *turtle geometry* [1]. A turtle, starting at a specified location and orientation in world-space, interprets an L-string as a series of position- and orientation-changing instructions. The position of the turtle is represented by a vector $\boldsymbol{p}$; its orientation is given by three vectors $\boldsymbol{h}$, $\boldsymbol{l}$, and $\boldsymbol{u}$, indicating in the turtle's local frame of reference which directions are forward (or *heading*), *left*, and *up*. As the L-string is scanned from left to right, these four state vectors change according to the instructions encoded in the string. In addition to the basic symbols listed in Table 1, many additional symbols encode the information required for a detailed, realistic plant model. For a thorough introduction to plant modeling using L-systems see *The Algorithmic Beauty of Plants* [11].

$F(a)$	move forward distance a
$+(\theta)$	turn left θ degrees
$-(\theta)$	turn right θ degrees
$\&(\theta)$	pitch down θ degrees
$\wedge(\theta)$	pitch up θ degrees
$\backslash(\theta)$	roll right θ degrees
$/(\theta)$	roll left θ degrees
[	save state, start new branch
]	end branch, restore state

Table 1 Basic L-string symbols and their turtle interpretations.

In addition, in the context of this work we introduce a new symbol $Z(s^h, s^u, s^l)$ to specify the stiffness of joints between subsequent branch segments. The impact of the parameters s^h, s^u, and s^l is discussed in Section 3.1. If the stiffness values are not incorporated in the L-system model and the resulting L-string, *ilsa* estimates them for the purpose of model manipulation, as described in Section 3.3.

3 Arrangement

Our interactive application *ilsa* acts as a specialized editor of L-strings. To allow interactive response time, *ilsa* displays an undetailed view of the model being manipulated. A detailed view is displayed in *cpfg* (continuous plant and fractal generator), the L-system interpretation and rendering environment developed by Prusinkiewicz *et al.* [4]. This detailed view is updated as arrangement changes are made in *ilsa*. The arranged model preserves the growth and environmentally-sensitive behavior of the original model. An example of the two applications working together is shown in Figure 1.

Since our goal is to preserve and augment the ability of L-systems to model botanically-accurate plants, we constrain the manipulations allowed by *ilsa* to those that could be performed on an actual plant: bending and pruning branches and arranging and clipping leaves and flowers. Removing elements (i.e., pruning or clipping them) is straightforward and will not be discussed in depth. Bending branches interactively in a physically plausible way is a much more interesting problem and will be fully explored in this section.

In nature, branches are continuous, beam-like structures that bend and twist in response to natural forces. The degree to which a branch bends or twists depends on its *stiffness*. In *ilsa*, branches are chains of straight, rigid segments connected by spring-like joints that bend and twist in response to user input. Consider a branch with n segments and n joints. At each joint i, numbered from 1 to n, the vectors $\boldsymbol{h_i}$, $\boldsymbol{u_i}$, and $\boldsymbol{l_i}$ define the *heading*, *up*, and *left* axes of the turtle reference frame of the next segment. These vectors are illustrated in Figure 2. Each joint i has three associated stiffness values: s_h, s_u, and s_l that define how difficult it is to rotate the joint about the local $\boldsymbol{h}$, $\boldsymbol{u}$, and $\boldsymbol{l}$ axes. Meaningful stiffness values range from 0 to infinity. An infinitely stiff joint allows no rotation around the axis in question, and a joint with zero stiffness rotates freely. When a plant is manipulated in *ilsa*, joint stiffness information is used by an inverse-kinematics (IK) optimization to determine a "natural" branch position.

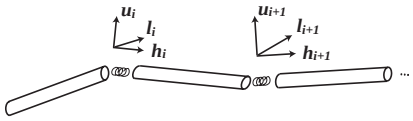


Figure 2 Branches in *ilsa* are made up of rigid segments connected by spring-like joints. At each joint i, the vectors $\boldsymbol{h_i}$, $\boldsymbol{u_i}$, and $\boldsymbol{l_i}$ define the heading, up, and left vectors of the turtle as it draws the next segment.

3.1 Branch manipulation

Manipulation in *ilsa* must satisfy two goals: it must seem realistic, and it must be interactive. A physical simulation would satisfy the first goal but would be too slow for our purposes. Instead, we use flexural and torsional joint stiffness values in an IK optimization that allows interactive manipulation and achieves pleasing, "natural" branch arrangements.

The branch-positioning process in *ilsa* consists of two nested loops. As the user manipulates a branch, an outer loop repeatedly calls the IK optimization routine with parameters supplied by the user interaction. Each time the optimization routine is called, it solves for a new branch position based on the current position of the branch, the supplied user input, and the stiffness of the branch. First we will discuss the objective function used by the inner loop of the optimization; we will then discuss the parameters passed to the optimization routine by the outer loop.

Objective function

The objective function minimized by *ilsa*'s IK solver positions a branch by solving for a set of joint rotations along that branch. Examining the objective function for a series of n different joints with one degree of freedom (hereafter referred to as *1-joints*), let i be an integer from 1 to n, and let α_i be the current angle at joint i. We refer to the joint angles collectively as the vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$. When manipulating a branch, the user is primarily interested in three things: the location of the end of the branch, the twist of the branch, and the curve of the branch. Our objective function contains two terms that attempt to meet user-supplied targets for branch end-position and branch twist, and two terms that attempt to maintain "natural" branch curvature. The four terms are described below:

- ***branch end-position***

  The branch end-position term tries to match the current branch end-position, denoted by $\boldsymbol{p}$, with the target branch end-position, denoted by $\hat{\boldsymbol{p}}$. The branch end-position term is then $\|\boldsymbol{p} - \hat{\boldsymbol{p}}\|$.

- ***branch twist***

  The branch twist term considers the vectors $\boldsymbol{l}$ and $\boldsymbol{u}$ of the final segment in the manipulated branch. We call these the *twist vectors* and denote them by $\boldsymbol{s}$ and $\boldsymbol{t}$ respectively. The twist term tries to minimize the difference between the current twist vectors and the target twist vectors, denoted by $\hat{\boldsymbol{s}}$ and $\hat{\boldsymbol{t}}$. The branch twist term is then $\|\boldsymbol{s} - \hat{\boldsymbol{s}}\| + \|\boldsymbol{t} - \hat{\boldsymbol{t}}\|$.

- ***spring-energy***

  The spring-energy term is a sum over the n joints in the branch and effectively bends flexible parts of the branch more than stiff parts. Let s_i be the stiffness value at joint i, where i is an integer from 1 to n. Let α_i be the current angle at joint i, let ρ_i be the rest angle at joint i, and define δ_i to be $\alpha_i - \rho_i$. The spring-energy term is then $\sum_{i=1}^{n} s_i \delta_i^2$. Examining the spring-energy term of a joint i in terms of the potential energy of a spring $\frac{1}{2}kx^2$, we see that the spring constant k represents the stiffness of the joint.

- ***smoothness***

  The smoothness term is a heuristic term designed to provide reasonable behavior when the input L-system does not contain joint stiffness information. This term attempts to minimize the difference in bend between $n - 1$ consecutive joints. The smoothness term is $\sum_{i=1}^{n-1} (\delta_{i+1} - \delta_i)^2$. When joint stiffness information is present in the input L-system, the user turns off smoothness by setting the importance weight of this term to 0.

Let w_p, w_t, w_e, and w_s denote the importance weights for the branch end-position term, the branch twist term, the spring-energy term, and the smoothness term respectively. The objective function for a set of n 1-joints is then the weighted sum of the four terms described above:

$$f(\boldsymbol{\alpha}) = w_p \|\boldsymbol{p} - \hat{\boldsymbol{p}}\| + w_t (\|\boldsymbol{s} - \hat{\boldsymbol{s}}\| + \|\boldsymbol{t} - \hat{\boldsymbol{t}}\|) + w_e \sum_{i=1}^{n} s_i \delta_i^2 + w_s \sum_{i=1}^{n-1} (\delta_{i+1} - \delta_i)^2 \quad (1)$$

For a three-dimensional plant model, each branch joint has three degrees of freedom. (We will call such a joint a *3-joint*). To arrange

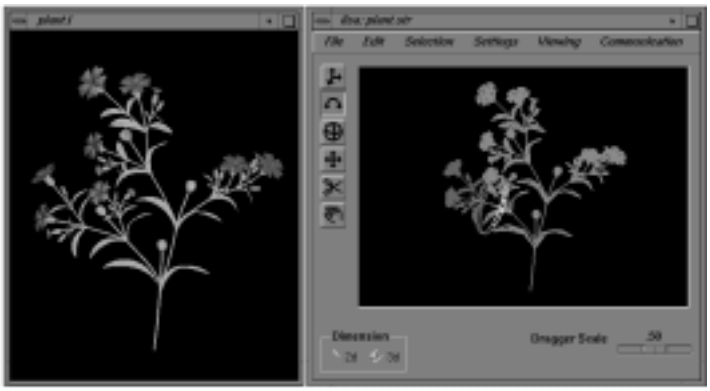
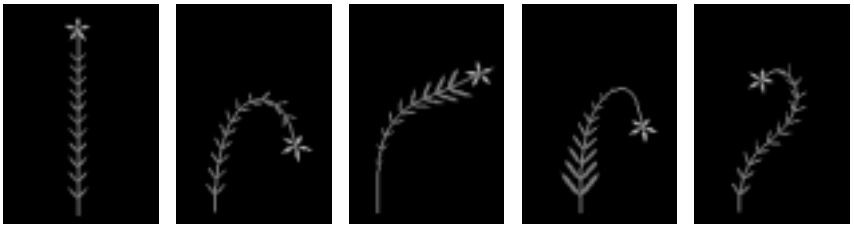


Figure 1 Our application *ilsa* was designed to work in conjunction with *cpfg*. The detailed rendering on the left is provided by *cpfg*; *ilsa* uses a simpler rendering style to allow interactive response time.



(a) (b) (c) (d) (e)

Figure 3 Branch manipulation in 2D. The branch consists of 11 segments connected by 1-joints. From left to right: (a) the original branch; (b) the result of manipulating a branch with constant joint stiffness values; (c) the result of manipulating a branch with increasing joint stiffness values; (d) the result of manipulating a branch with decreasing joint stiffness values; and (e) the result of manipulating the uniformly stiff branch in two separate moves. Leaf sizes indicate the stiffness of the associated joints.

a branch in three-dimensions, *ilsa* interprets each 3-joint as three 1-joints, two of which are followed by a segment of length zero.

The importance of the spring-energy term to the results of a manipulation is illustrated in Figures 3 and 4. In Figure 3 we show the results of manipulating simple branches with constant, increasing, and decreasing joint stiffness values. In Figure 4 we show the results of applying a single manipulation to a more complex branch in three situations: first using no stiffness information and no smoothness term, next using only the smoothness term, and finally using only stiffness information. The same manipulation was used in each case: the left branch was selected just above the last joint and pulled down and to the left. The difference between subfigures (b) and (c) indicates that in the absence of stiffness information, using the smoothness term gives slightly more natural results. Obtaining the most natural result, however, requires stiffness values.

Optimization parameters

In *ilsa*, we use conjugate gradient descent optimization, since the function we are optimizing is nonlinear and differentiable [9]. (For details on the derivative function, see the appendix.) Conjugate gradient descent is sensitive to local minima. However, this potential shortcoming does not present serious problems in practice because our application is interactive and because the optimization converges quickly given a good starting point. Thus, any branch that settles into an undesired configuration due to a local minimum can simply be "pulled out" interactively, and the optimization will continue. This approach is also used by Zhao and Badler in their work on positioning articulate figures using inverse kinematics [15].

Positioning a branch requires the specification of a set of n joints with joint angle values $\boldsymbol{\alpha}$. Let the initial joint angle values be $\bar{\boldsymbol{\alpha}}$. A new set of joint angle values $\boldsymbol{\alpha}$ is chosen by the IK-solver using $\bar{\boldsymbol{\alpha}}$ as a starting point. The specified set of n joints defines a set of branch segments that will be directly affected by the manipulation. We call this set of branch segments the *branch path*. Let $\boldsymbol{r}$ denote the position of the root of the branch path and $\boldsymbol{p}$ denote the position of the movable end. (Note that the position of $\boldsymbol{r}$ will not change during manipulation.) The *branch axis-vector* $\boldsymbol{v}(\boldsymbol{p}, \boldsymbol{r})$ describes a vector pointing from the root of the branch path to its movable end. As we described in the previous section, the *twist vectors* $\boldsymbol{s}$ and $\boldsymbol{t}$ are the vectors $\boldsymbol{l}$ and $\boldsymbol{u}$ respectively of the last joint in the branch path. A branch path and its associated vectors are illustrated in Figure 5.

As the user interacts with a branch in *ilsa*, an outer loop repeatedly calls the IK-optimization routine with targets for the branch end-position and twist-vectors $\hat{\boldsymbol{p}}$, $\hat{\boldsymbol{s}}$, and $\hat{\boldsymbol{t}}$, respectively. The optimization routine solves for a new set of joint angle values $\boldsymbol{\alpha}$. The actual end-position and twist-vectors of the branch change as a function of the new joint angles; that is, $\boldsymbol{p}$, $\boldsymbol{s}$ and $\boldsymbol{t}$ are functions of $\boldsymbol{\alpha}$.

The inner loop of the optimization process minimizes the objective function $f(\boldsymbol{\alpha})$. The branch end-position term of the objective function requires $\boldsymbol{p}(\boldsymbol{\alpha})$, which is calculated, and the target branch end-position $\hat{\boldsymbol{p}}$, which is supplied as input. The branch twist term of the objective function requires the twist vectors $\boldsymbol{s}(\boldsymbol{\alpha})$ and $\boldsymbol{t}(\boldsymbol{\alpha})$, which are calculated, and the target twist vectors $\hat{\boldsymbol{s}}$ and $\hat{\boldsymbol{t}}$, which are supplied as input. Thus the full set of inputs to the optimization routine is $\bar{\boldsymbol{\alpha}}$, $\hat{\boldsymbol{p}}$, $\boldsymbol{r}$, $\hat{\boldsymbol{t}}$, and $\hat{\boldsymbol{s}}$. The values of $\boldsymbol{\alpha}$ are determined by conjugate gradient descent from a starting set of values $\bar{\boldsymbol{\alpha}}$.

As the following section will describe, *ilsa* provides manipulation widgets that allow subsets of the optimization parameters to vary simultaneously, while holding other parameters fixed. These widgets allow the user to control which aspects of a branch will be affected by a manipulation and interact with a plant in meaningful ways by rotating, twisting, straightening, and bending branches.

(a) (b) (c) (d)

Figure 4 Plant manipulation with and without joint stiffness information. From left to right: (a) the original plant with the active branch in dark gray and the branch path in white; (b) the result of manipulating the lower branch with no joint stiffness values and smoothness turned off; (c) the result of manipulating the branch with no joint stiffness values and smoothness turned on; (d) the result of manipulating the branch with automatically-assigned joint stiffness values and smoothness turned off.

3.2 User interface

As mentioned before, *ilsa* renders models using lines for branches and wire-frame surfaces for leaves and petals. This drawing style allows *ilsa* to achieve good interactive behavior even for complex models. A detailed view of the plant is provided by *cpfg* and updated in response to messages sent by *ilsa* through a socket connection. The manipulations allowed by *ilsa* include pruning branches, clipping leaves and flowers, and arranging branches by bending and twisting them.

Pruning

Pruning and clipping are straightforward: the user simply chooses the delete tool, selects the portion of the plant to be removed, and hits delete on the keyboard. A special symbol is inserted into the L-string to indicate that pruning has taken place. This symbol can be used in further generations of the L-system model to trigger regrowth induced by pruning [10].

Selection

In order to arrange branches in *ilsa*, the user first selects the part of the plant to be manipulated. This selection process has two components, as does the analogous selection process on a real plant. Imagine that a user wants to physically bend an inconveniently-placed branch. She holds the base of the plant in her left hand, then grasps the end of the plant in her right hand, and pulls. The manipulation directly affects a path of branch segments, starting with the *manipulated point* on the branch that she pulls and ending at the *fixed point* at the base of the plant. Now imagine that the user's goal is to achieve a more local change. She chooses a manipulated point at the end of the branch as before, but she chooses a different fixed point, one above the base of the plant, shortening the affected path. In either case, the effect of the manipulation may or may not propagate all the way from the manipulated point to the fixed point, depending on the rigidity of the branches along the way.

We have already defined the set of branch segments that may be directly affected by a manipulation as the *branch path*. There may be any number of branches attaching to the branch path whose positions may also be indirectly affected by this manipulation. We call this set of branches the *active branch*. An example of an active branch is shown in Figure 6. To define a branch path and an active branch in *ilsa*, the user first picks one point to be the fixed point of both the branch path and the active branch, the point $\boldsymbol{r}$ in our previous discussion. The active branch consists of all structures from $\boldsymbol{r}$ to the branch ends, and these structures are highlighted as soon as $\boldsymbol{r}$ is picked. The active branch selection can be easily modified using the arrow keys: up moves the fixed point toward the branch end, down moves the fixed point toward the base of the plant, and left and right move to sibling branches. The user specifies the moveable end of the branch path, the manipulated point $\boldsymbol{p}$, by picking again within the active branch.

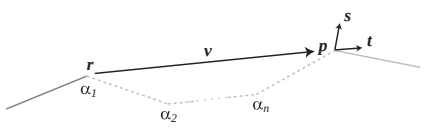


Figure 5 Branch path. The branch path, shown as dashed lines, consists of n segments between the branch path root $\boldsymbol{r}$ and the branch path end $\boldsymbol{p}$. The branch axis vector $\boldsymbol{v}$ and the twist vectors $\boldsymbol{s}$ and $\boldsymbol{t}$ are used by *ilsa*'s IK-optimization over joint angles α_i, $1 \leq i \leq n$.

Figure 6 In *ilsa*, the *active branch* is highlighted in red (shown in dark gray), the *branch path* is highlighted in yellow (shown in white), and the immobile portion of the plant remains green (shown in middle gray).

Arrangement

Once the branch path has been selected, an arrangement tool appears at the end of the branch path. The four arrangement tools that *ilsa* provides come from the Open Inventor Toolkit and are shown in Figure 7 [13]. These tools allow the user to modify different subsets of the parameters passed to the optimization routine. Any parameter that is unaffected by a particular tool remains fixed at the value the parameter had at the time the arrangement tool was instantiated. Let $\bar{\boldsymbol{p}}$, $\bar{\boldsymbol{s}}$, and $\bar{\boldsymbol{t}}$ denote these *original values* of $\boldsymbol{p}$, $\boldsymbol{s}$, and $\boldsymbol{t}$ respectively. The optimization parameters that vary according to arrangement tool are $\hat{\boldsymbol{p}}$ (target branch end-position) and $\hat{\boldsymbol{s}}$ and $\hat{\boldsymbol{t}}$ (target twist-vectors). Let $\bar{\boldsymbol{v}} = \bar{\boldsymbol{p}} - \boldsymbol{r}$ denote the original branch axis-vector. The behaviors of the arrangement tools and their interaction with the optimization routine are described below:

- ***Spherical dragger***

  A spherical widget centered at $\boldsymbol{r}$ with a radius $\|\bar{\boldsymbol{v}}\|$ allows the user to rotate and drag the branch with track-ball-like interactions. The spherical dragger affects all three parameters: $\hat{\boldsymbol{p}}$, $\hat{\boldsymbol{s}}$, and $\hat{\boldsymbol{t}}$.

(a) (b) (c) (d)

Figure 7 Arrangement widgets. From left to right: the spherical dragger, disc dragger, axis dragger, and 2D-plane dragger.

- ***Disc dragger***

  A disc-shaped widget positioned at $\bar{\boldsymbol{p}}$ and axis-aligned with $\bar{\boldsymbol{v}}$ allows the user to twist the branch with steering-wheel-like motions. The disc dragger affects the branch twist parameters: target twist vectors $\hat{\boldsymbol{s}}$ and $\hat{\boldsymbol{t}}$. Since the branch end-position should stay fixed, the dragger always passes $\bar{\boldsymbol{p}}$ as the target branch end-position $\hat{\boldsymbol{p}}$.

- ***Axis dragger***

  A widget consisting of a cylinder and an orthogonal plane positioned at $\bar{\boldsymbol{p}}$. The cylinder is axis-aligned with $\bar{\boldsymbol{v}}$ and allows the user to move the branch end-position in and out along $\bar{\boldsymbol{v}}$. (The user may also use the plane portion of the widget to position the end of the branch, but we have found that the spherical dragger provides a more natural interface for this action, since it attempts to preserve the length of the branch.) The axis dragger affects only the target branch end-position parameter $\hat{\boldsymbol{p}}$. Since branch twist does not change, the dragger always passes the initial twist vectors $\bar{\boldsymbol{s}}$ and $\bar{\boldsymbol{t}}$ for the target twist vectors $\hat{\boldsymbol{s}}$ and $\hat{\boldsymbol{t}}$.

- ***2D-plane dragger***

  This dragger consists of the axis dragger widget, positioned so that the plane is useful. The dragger appears at $\bar{\boldsymbol{p}}$ and is oriented in the view-plane. It allows the user to interact with a 3-dimensional plant as if it were 2-dimensional. The dragger affects only the target branch end-position parameter $\hat{\boldsymbol{p}}$ only. The dragger always passes the initial twist vectors $\bar{\boldsymbol{s}}$ and $\boldsymbol{t}$ for the target twist vectors $\hat{\boldsymbol{s}}$ and $\hat{\boldsymbol{t}}$.

The spherical dragger can also be used to manipulate structures such as leaves and flowers. In this case, the root of the branch path $\boldsymbol{r}$ is the attachment point of the structure, and the branch path consists of a single virtual segment in the direction of the turtle heading vector $\boldsymbol{h}$ at $\boldsymbol{r}$. The length of the virtual segment is chosen such that $\|\bar{\boldsymbol{v}}\|$ equals the maximum dimension of the selected structure.

3.3 Assigning joint stiffness values

Branch stiffness values are not readily available for most plants, nor are they easily measured. Therefore, we want our system to automatically assign reasonable joint stiffness values to existing L-system models. Studies in the field of plant biomechanics, along with some help from a Renaissance man, provide the tools we need to accomplish this task. As discussed in detail by Niklas, plant branches behave like beams that are subject to *flexion*, or bending, and *torsion*, or twisting [8]. The flexural and torsional rigidities of a branch depend on both its material and cross-sectional geometry. If we assume that all branches of a plant are composed of the same material and have circular cross-sections, we can determine relative branch rigidities based solely on branch radii.

Branch flexion

In turtle terms, branch flexion means rotation about the $\boldsymbol{l}$ or $\boldsymbol{u}$ axis. Our goal is therefore to obtain values for s_i^l and s_i^u for each joint i of a branch.

Borrowing notation from the field of biomechanics, let E represent the *elastic modulus* of the branch material and let I represent the *second moment of area* of the branch cross-section.[1] Let the magnitude of the torque required to bend the branch to a curvature K be represented by τ_f. Then, according to Niklas [8, page 135],

$$\tau_f = EIK. \tag{2}$$

We see that the torque required to bend a branch to a curvature K depends on some inherent properties of the branch and on the degree of curvature. The resistance of the branch to bending, represented by the product of E and I, is called the *flexural stiffness* of the branch [8].

Typically we do not know the value of E, the elastic modulus of the branch material. However, using our assumptions that E is constant and that the branch has a circular cross-section of radius r, we can conclude that the flexural stiffness EI is proportional to r^4 [8, page 134]. This information is sufficient for our purposes, since the multiplication of all the joint stiffness terms by a constant does not change the outcome of the branch-positioning optimization. Let us denote by c_f the constant portion of flexural stiffness:

$$EI \equiv c_f r^4.$$

Equation (2) describes the behavior of a continuous branch, but in L-system models a branch is represented as a sequence of stiff segments connected by joints. We can replace curvature by its discrete approximation: $K = \theta/L$, where $L = (l_{i-1} + l_i)/2$ is the average length of two adjacent segments, and θ is the angle between them. By substituting this expression into Equation (2), we obtain:

$$\tau_f = \frac{c_f r^4 \theta}{L} = \frac{2 c_f r^4 \theta}{l_{i-1} + l_i}. \tag{3}$$

Not surprisingly, we see that the torque required to bend a joint to an angle θ is proportional to θ.

Since we have assumed circular cross-section of the branch, the flexural stiffness values for bending around the u and l axes are the same:

$$s_i^u = s_i^l = \frac{2 c_f r^4}{\ell_{i-1} + \ell_i}. \tag{4}$$

[1]The second moment of area describes both the geometry of the cross-section and the plane of bending. In a bending branch, there is a *neutral axis* at which opposing tensile (stretching) and compressive stresses balance. The second moment of area, I, is an integral summing the products of each infinitesimally small area within a cross-section and the square of the distance d each area lies from the neutral axis: $I = \int_{-d_{max}}^{d_{max}} d^2 dA$. For example, for a branch with a circular cross-section of radius r, the second moment of area $I = \frac{1}{4}\pi r^4$ [8, page 134].

Equation (4) allows us to assign flexural stiffness values to the joints of a branch once we know the flexural constant c_f for the plant and the radius r of the branch in question.

Branch torsion

In *ilsa*, torsion corresponds to rotation around the ***h*** axis. To calculate s_i^h for each joint i of a branch, we consider the resistance of the branch to twisting. This resistance is the *torsional rigidity* of the branch. Like flexural stiffness, torsional rigidity depends both on the branch material and the branch cross-sectional area [8, page 160]. For a branch with a circular cross-section and constant material, there is a constant c_t for which the torsional rigidity S_t is given by

$$S_t \equiv \frac{c_t r^4}{L}.$$

Replacing L by the average length of the adjacent segments, we arrive at an equation for the torsional rigidity of the i^{th} joint of a particular branch with radius r:

$$s_i^h = \frac{2c_t r^4}{\ell_{i-1} + \ell_i}. \tag{5}$$

Equation (5) allows us to assign torsional rigidity values to the joints of a branch once we know the torsional constant c_t for the plant and the radius r of the branch in question.

For branches with non-circular cross-sections, we can calculate two second moments of area, I^l and I^u, to account for differences in flexibility along the ***l*** and ***u*** axes.

Automatic assignment

It should now be clear that given values for the flexural stiffness constant c_f, the torsional rigidity constant c_t, and the radii of all the branches in the plant, we can automatically assign joint stiffness values.

If the radii values are not available, we assign reasonable values using a formula proposed by Leonardo da Vinci. He postulated that the cross-sectional area of a tree branch is equal to the sum of the cross-sectional areas of its children branches [2].

Let us assume that all children of a particular branch have the same radius. Let r represent the radius of a branch with n children and let r_c represent the radius of the children. Using the relationship proposed by da Vinci, we derive the following equations:

$$\pi r^2 = n\pi r_c^2 \Rightarrow r_c^2 = \frac{r^2}{n}. \tag{6}$$

In *ilsa*, the user provides a value for the trunk radius r_0 (in terms of turtle steps). The system then recursively assigns radii values to all branches using Equation (6).

The user also provides values for the flexural stiffness and torsional rigidity of the trunk, s_0^u (or s_0^l) and s_0^h respectively. These values represent the stiffness of a joint between segments with average length equal to 1. We then use Equations (4) and (5) to calculate the constants c_f and c_t:

$$c_f = \frac{s_0^u}{{r_0}^4},$$
$$c_t = \frac{s_0^h}{{r_0}^4}.$$

The following pseudocode summarizes the recursive function used to automatically assign joint stiffness values to a plant:

function *AssignStiffness*(*branch*, r, c_f, c_t)
  for each joint i
    $s_i^u := \frac{2c_f r^4}{\ell_{i-1}+\ell_i}$
    $s_i^l := s_i^u$
    $s_i^h := \frac{2c_t r^4}{\ell_{i-1}+\ell_i}$
  end for
  ${r_c}^2 := \frac{r^2}{n}$
  for each child branch *child*
    AssignStiffness(*child*, r_c, c_f, c_t)
end function

4 Examples

We present three examples of plants generated using *cpfg* and arranged using *ilsa*:

- ***Rose campion***

  We selected the rose campion (*Lynchnis coronaria*) because its architecture has already been described and modeled in detail [11, 12]. Figure 8 presents three snapshots documenting the arrangement process. From the complex model generated by *cpfg*, we pruned and arranged the branches to create a simpler, more stylized plant. This arrangement was completed in about half an hour.

- ***Lily***

  The lily model depicted in Figure 9 has exquisite detail, much of which is hidden in the original arrangement as generated by *cpfg*. The original model did not provide branch stiffness information, so we assigned joint stiffness values in *ilsa*. This arrangement took about an hour to complete. A great deal of time was spent waiting for *cpfg* to render the detailed version of the plant so that the effects of precise manipulations might be discerned.

- ***Regrowth example***

  The model depicted in Figure 10 is a very simple plant, based on an example from *The Algorithmic Beauty of Plants.* The branches are rendered as lines and the leaves and flowers as polygons. Joint stiffness values were again assigned using *ilsa*. This example hints at the many possible applications of the ability to grow a plant, arrange it, and then grow it some more.

5 Discussion and future work

We have presented a system for the interactive manipulation of plants modeled using L-systems. Our application *ilsa* allows the user to locally arrange a plant model while preserving its botanical accuracy. The approach implemented in *ilsa* is to interact with an L-system model at the level of the L-string.

This work extends the usefulness of L-systems for both computer graphics and botanical modeling. Possible applications include L-system bonsai, interactive topiary, landscaping planning and design. Our system could easily be extended to support flower arranging and grafting, a process widely applied to fruit trees. Illustrators, scene designers, and animators could take advantage of

the growth and environmental interaction capabilities of L-system models, yet achieve more precise control over the final presentation of the plants. Another interesting area for further exploration is 2D floral pattern design, potentially extending the work of Wong *et al.* [14].

Another very interesting direction for future work is the development of an interactive toolkit for creating L-system models. We imagine a library of basic components like branching structures, leaf profiles and surfaces, flowering patterns, thorns, etc. that could be put together and interactively adjusted, resulting in a new model. Lintermann *et al.* developed a system that allows intuitive assembly of botanical components [6]. Extending the user-interface concepts presented in their work to take advantage of the growth and environmental interaction capabilities of L-system models could lead to a powerful botanical modeling system with an easy-to-use visual interface.

Acknowledgments

Many thanks to the people who volunteered their time and expertise to help with this project, especially Radomír Mech and Eric Stollnitz. This work was supported by an NSF Presidential Faculty Fellow award (CCR-9553199), an ONR Young Investigator award (N00014-95-1-0728), a NASA Space Grant, an NSF Graduate Research Fellowship, and industrial gifts from Interval, Microsoft, and Xerox to David Salesin, and the NSERC grant OGP0130084 to Przemyslaw Prusinkiewicz.

Appendix: Gradient of objective function

Here we derive the gradient of the objective function used in the IK-optimization and presented in Equation (1):

$$\begin{aligned} f(\boldsymbol{\alpha}) &= w_p\|\boldsymbol{p}-\hat{\boldsymbol{p}}\| + w_t(\|\boldsymbol{s}-\hat{\boldsymbol{s}}\| + \|\boldsymbol{t}-\hat{\boldsymbol{t}}\|) \\ &\quad + w_e\sum_{i=1}^{n} s_i\delta_i^2 + w_s\sum_{i=1}^{n-1}(\delta_{i+1}-\delta_i)^2 \end{aligned}$$

The gradient is a vector of partial derivatives:

$$\nabla f(\boldsymbol{\alpha}) = \left(\frac{\partial f}{\partial\alpha_1}, \frac{\partial f}{\partial\alpha_2}, \ldots, \frac{\partial f}{\partial\alpha_n}\right)$$

We need to find the partial derivative of f with respect to a single joint rotation α_i:

$$\begin{aligned} \frac{\partial f}{\partial\alpha_i} &= \frac{\partial}{\partial\alpha_i}w_p(\|\boldsymbol{p}-\hat{\boldsymbol{p}}\|) + \\ &\quad \frac{\partial}{\partial\alpha_i}w_t(\|\boldsymbol{s}-\hat{\boldsymbol{s}}\| + \|\boldsymbol{t}-\hat{\boldsymbol{t}}\|) + \\ &\quad \frac{\partial}{\partial\alpha_i}w_e s_i\delta_i^2 + \\ &\quad \frac{\partial}{\partial\alpha_i}w_s((\delta_{i+1}-\delta_i)^2 + (\delta_i-\delta_{i-1})^2) \end{aligned}$$

For clarity we consider the derivative terms one at a time. Let A, B, C, and D be the derivatives of the branch end-position term, the branch twist term, the spring-energy term, and the smoothness term respectively:

$$\frac{\partial f}{\partial\alpha_i} = A + B + C + D. \tag{7}$$

Let us first look at the partial derivative of the branch end-position term with respect to α_i. Let the rotation axis of the i^{th} joint be the unit vector $\boldsymbol{w}$. Let $\boldsymbol{p}_i$ denote the position of the turtle at joint i, after drawing segment $i-1$. Let the branch end-position $\boldsymbol{p} = \boldsymbol{p}_n$. Let $\boldsymbol{v}_{post} = \boldsymbol{p} - \boldsymbol{p}_i$. Let $\boldsymbol{s}$ and $\boldsymbol{t}$ be the turtle vectors $\boldsymbol{l}$ and $\boldsymbol{u}$ respectively after drawing the n^{th} branch segment. The derivatives of $\boldsymbol{p}$, $\boldsymbol{s}$, and $\boldsymbol{t}$ can be computed from $\boldsymbol{w}$ and $\boldsymbol{v}_{post}$ as follows [15]:

$$\begin{aligned} \frac{\partial\boldsymbol{p}}{\partial\alpha_i} &= \boldsymbol{w}\times\boldsymbol{v}_{post} \\ \frac{\partial\boldsymbol{s}}{\partial\alpha_i} &= \boldsymbol{w}\times\boldsymbol{s} \\ \frac{\partial\boldsymbol{t}}{\partial\alpha_i} &= \boldsymbol{w}\times\boldsymbol{t} \end{aligned}$$

Thus,

$$\begin{aligned} A &= \frac{\partial}{\partial\alpha_i}w_p(\|\boldsymbol{p}-\hat{\boldsymbol{p}}\|) \\ &= 2w_p(\boldsymbol{p}-\hat{\boldsymbol{p}})\cdot\frac{\partial\boldsymbol{p}}{\partial\alpha_i} \end{aligned} \tag{8}$$

Now let us look at the partial derivative of the branch twist-term with respect to α_i:

$$\begin{aligned} B &= \frac{\partial}{\partial\alpha_i}w_t(\|\boldsymbol{s}-\hat{\boldsymbol{s}}\| + \|\boldsymbol{t}-\hat{\boldsymbol{t}}\|) \\ &= 2w_t((\boldsymbol{s}-\hat{\boldsymbol{s}})\cdot\frac{\partial\boldsymbol{s}}{\partial\alpha_i} + (\boldsymbol{t}-\hat{\boldsymbol{t}})\cdot\frac{\partial\boldsymbol{t}}{\partial\alpha_i}) \end{aligned} \tag{9}$$

Now let us look at the partial derivatives of the branch energy and smoothness terms with respect to α_i:

$$\begin{aligned} C &= \frac{\partial}{\partial\alpha_i}w_e s_i\delta_i^2 \\ &= w_e s_i\delta_i \end{aligned} \tag{10}$$

and

$$\begin{aligned} D &= \frac{\partial}{\partial\alpha_i}w_s((\delta_{i+1}-\delta_i)^2 + (\delta_i-\delta_{i-1})^2) \\ &= w_s(2\delta_i - \delta_{i-1} - \delta_{i+1}) \end{aligned} \tag{11}$$

By substituting expressions (2) to (5) into (1), we obtain the gradient function needed.

References

[1] H. Abelson and A. A. diSessa. *Turtle geometry*. M.I.T. Press, Cambridge, 1982.

[2] L. da Vinci. *The notebooks of Leonardo da Vinci, compiled and edited from the original manuscripts by Jean Paul Richter*. Dover Publications, New York, 1970.

[3] Mikako Harada, Andrew Witkin, and David Baraff. Interactive physically-based manipulation of discrete/continuous models. In *SIGGRAPH 95 Conference Proceedings*, pages 199–208. ACM SIGGRAPH, New York, 1995.

[4] Mark James, Mark Hammal, Jim Hanan, Radomír Mech, and Przemyslaw Prusinkiewicz. *CPFG Version 2.7 User's Manual*.

[5] A. Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[6] Berd Lintermann and Oliver Deussen. Interactive modelling and animation of branching botanical structures. In *Eurographics Computer Animation and Simulation EGCAS96*. Springer-Verlag, 1996.

[7] Radomír Mech and Przemyslaw Prusinkiewicz. Visual models of plants interacting with their environment. In *SIGGRAPH 96 Conference Proceedings*, pages 397–410. ACM SIGGRAPH, New York, 1996.

[8] K. J. Niklas. *Plant Biomechanics: an Engineering Approach to Plant Form and Function*. The University of Chicago Press, Chicago, 1992.

[9] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*, chapter Chapter 10: Minimization or Maximization of Functions. Cambridge University Press, 1992.

[10] Przemyslaw Prusinkiewicz, Mark James, and Radomír Mech. Synthetic topiary. In *SIGGRAPH 94 Conference Proceedings*, pages 351–358. ACM SIGGRAPH, New York, 1994.

[11] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[12] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and James Hanan. Developmental models of herbaceous plants for computer imagery purposes. In *SIGGRAPH 88 Conference Proceedings*, pages 141–150. ACM SIGGRAPH, New York, 1988.

[13] Josie Wernecke. *The Inventor Mentor*. Addison-Wesley Publishing Company, 1994.

[14] Michael T. Wong, Douglas E. Zongker, and David H. Salesin. Computer-generated floral ornament. In *SIGGRAPH 98 Conference Proceedings*, pages 423–434. ACM SIGGRAPH, New York, 1998.

[15] Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics*, 13(4):313–336, October 1994.

(a) (b) (c)

Figure 8 Rose campion. From left to right: (a) the original L-system model; (b) an intermediate stage in the arrangement process; (c) the final arrangement.

(a) (b)

Figure 9 Lily. The original L-system model (a) and the final arrangement (b).

(a) (b) (c) (d)

Figure 10 Regrowth example. From left to right: (a) a simple plant grown for four generations; (b) the same plant grown for five generations (rescaled to the same display size); (c) the fourth-generation plant arranged in *ilsa*; (d) the arranged plant grown for one more generation.