Combining Finite Element Method and L-Systems Using Natural Information Flow Propagation to Simulate Growing Dynamical Systems

Jean-Philippe Bernard[1], Benjamin Gilles[2], and Christophe Godin[1]

[1] Inria, Virtual Plants project-team, Université Montpellier 2, Bâtiment 5, CC 06002, 860 rue de Saint Priest, 34095 Montpellier Cedex 5, France
[2] CNRS, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, Université Montpellier 2, Bâtiment 5, CC 06002, 860 rue de Saint Priest, 34095 Montpellier Cedex 5, France

Abstract. This paper shows how to solve a system of differential equations controlling the development of a dynamical system based on finite element method and L-Systems. Our methods leads to solve a linear system of equations by propagating the flow of information throughout the structure of the developing system in a natural way. The method is illustrated on the growth of a branching system whose axes bend under their own weight.

1 Introduction

Plants are complex branching organisms that undergo continuous development throughout their lifetime. To understand the key processes that control this development, a new type of modeling approach, called Functional-Structural Plant Models (FSPM) [8, 19, 17], has been developed in the last two decades. FSPMs combine a detailed description the plant architecture (in terms of axes or stem units) and physiological processes that participate to the branching system development (photosynthesis, water/sugar/mineral element transport, carbon allocation, bud growth, hormonal transport and regulation, interaction with gravity, etc.).

To build FSPMs, L-systems [16] have emerged as a dominant paradigm to describe both the development plant branching systems in time and to model the different bio-physical processes of interest [14, 3]. L-systems make it possible to model the development of a plant by specifying rules of development for the different types of considered plant constituent in a declarative manner. At each time step, the language engine scans the constituents of the branching structure being computed and applies the developmental rule that corresponds to its type. Interestingly, at no moment the modeler needs to index the plant elements. As the rules are supposed to be local, it is sufficient in the rule specification to access the immediate neighbor components, for example referring in the rule to the predecessor and successor components of the current plant component.

The propagation of a signal from the basis of the plant to the top provides a good example of such a principle of locality. Let a plant be represented a bracketed string `I I [ I I ] I [ I ] I`. This string represents a branching structure containing 7 components, all of type `I` (note that the structure contains no indexing of the components). Two consecutive `I`'s represent two consecutive segments in the same axis of the plant, while a bracket indicates a branch inserted at the top of preceding left-hand component (Fig. 1). Then, let us imagine that the leftmost component in the string (at the plant basis) contains a signal $x = 1$, and that the signal x is set to 0 in all the other components. To propagate the signal in time through the plant body, one needs to define a simple local rule such as (in pseudo-code form):

```
I  -->  { if predecessor().x == 1 then current().x = 1
        } produce I
```

meaning that a `I` symbol should be transformed over one time step in a `I` symbol (produce statement) after having set its internal signal value x to 1 if the x signal of the predecessor components in the plant was itself at set at 1. This

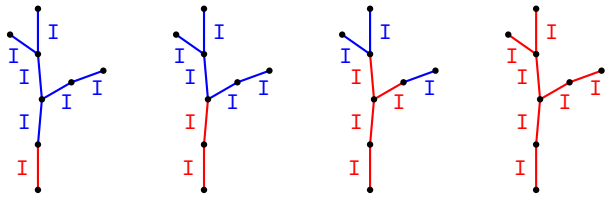


Fig. 1. Branch represented L-string `I I [ I I ] I [ I ] I` with information $x = 1$ (red segments) propagation to others segments (blue).

local rule makes it possible to get completely rid of indexes when transferring information through the plant structure [13]. This specific feature of L-systems was used in the last decade to develop computational models for which the flow of information propagates in a natural way over the plant structure from component to component, e.g. [1] for the transport of carbon, [15] for the transport of water, and [10, 5] for the reaction of plants to gravity. All these algorithms use finite difference methods (FDM) for which the plant is decomposed into a finite number of elements and quantities of interest (water content, sugar concentration, forces, displacements, etc.) correspond to discrete values attached to each component. Different FDM schemes have been developed for this based either on explicit or implicit methods [7, 9].

FDM approaches use a local Taylor expansion to approximate differential equations and are easy to implement. However, the quality of the approximation between grid points is generally considered poor. The Finite Element Method is an alternative solution that uses an integral formulation. While more complex to implement, the quality of a FEM approximation is often higher than in the corresponding FDM approach [11]. In this paper, we intend to adapt the

FEM approach to be used in the context of L-systems and natural computing, i.e. strictly respecting the paradigm of computational locality, and solving the differential equation by propagating information flows throughout the structure being computed. We illustrate and assess our generic approach on the problem of computing the effect of gravity on growing branching systems.

2 Natural Computing of Branch Bending Using Finite Difference Method (FDM) and L-Systems

2.1 Mechanical Model of Branch Bending

We model a branch as a set of inextensible elastic cantilever beams rigidly connected to each other and forming a branching network. Each beam represents a botanical axis and is conceptualized as a mean curve $\mathcal{C}$ of length L with natural parameter $s \in [0, L]$ denoting the arc-length distance of a point $\boldsymbol{P}(s)$ from the base of the stem and a section $\mathcal{S}(s)$ (Fig. 2a).

Each point $\boldsymbol{P}(s)$ is associated with an orthonormal local frame $\boldsymbol{\mathcal{R}}(s) = \{\boldsymbol{H}(s), \boldsymbol{L}(s), \boldsymbol{U}(s)\}$ (heading, left and up) similar to the Frenet's coordinate system [16]. We assume that vector $\boldsymbol{H}(s)$ is tangent to the rod axis and vectors $\boldsymbol{L}(s)$ and $\boldsymbol{U}(s)$ are $\mathcal{C}^0$-continuous with respect to s. Since all vectors $\boldsymbol{H}(s)$ have unit length the point $\boldsymbol{P}(s), s \in [0, L]$ is defined by:

$$\boldsymbol{P}(s) = \boldsymbol{P}(0) + \int_0^s \boldsymbol{H}(u)\,\mathrm{d}u. \tag{1}$$

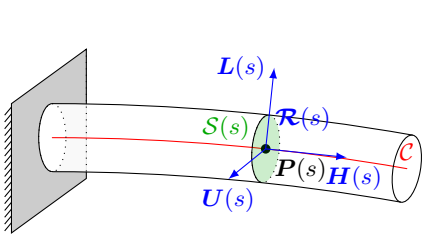
Let $\boldsymbol{P}(s)$ and $\boldsymbol{P}(s + ds)$ be two infinitesimally close points on the curve $\mathcal{C}$. Then the local frame $\boldsymbol{\mathcal{R}}(s + \mathrm{d}s)$ can be obtained from $\boldsymbol{\mathcal{R}}(s)$ by a rotation of axis $\Delta(s)$ and angle $\theta(s)$. It is convenient to represent this rotation by a vector $\boldsymbol{\Omega}(s)$, called the *generalized curvature*, whose direction is the rotation axis $\Delta(s)$ and whose norm is $\theta(s)$ (Fig. 2b) [10]. If the arc length $\mathrm{d}s$ is infinitesimal, this rotation can be factorized as a rotation around the tangent (twist) and a rotation around the normal (curvature) of the mean curve $\mathcal{C}$ at the point $\boldsymbol{P}(s)$. Starting from an initial frame $\boldsymbol{\mathcal{R}}(0)$, the frames $\boldsymbol{\mathcal{R}}(s)$ can be obtained thanks to the ordinary differential equation (2) [10]:

$$\mathrm{d}_s\boldsymbol{\mathcal{R}}(s) = [\boldsymbol{\Omega}(s)]_\times \boldsymbol{\mathcal{R}}(s), \tag{2}$$

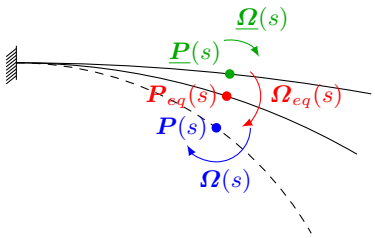
where $[\boldsymbol{\Omega}(s)]_\times$ denote respectively the skew-symmetric matrix corresponding to the cross product of $\boldsymbol{\Omega}(s)$ with an other vector (Eq. (3))and $\boldsymbol{\mathcal{R}}(s)$ denotes the column matrix $\big[\boldsymbol{H}(s), \boldsymbol{L}(s), \boldsymbol{U}(s)\big]$.

$$\boldsymbol{\Omega} \times \boldsymbol{v} = \begin{bmatrix} \boldsymbol{\Omega}_0 \\ \boldsymbol{\Omega}_1 \\ \boldsymbol{\Omega}_2 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{v}_0 \\ \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \end{bmatrix} = \begin{bmatrix} 0 & -\boldsymbol{\Omega}_2 & \boldsymbol{\Omega}_1 \\ \boldsymbol{\Omega}_2 & 0 & -\boldsymbol{\Omega}_0 \\ -\boldsymbol{\Omega}_1 & \boldsymbol{\Omega}_0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_0 \\ \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \end{bmatrix} = [\boldsymbol{\Omega}]_\times \boldsymbol{v} \tag{3}$$

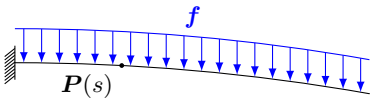
At rest, the branch geometry is characterized by its generalized curvature $\underline{\boldsymbol{\Omega}}$ and defines the reference configuration. At each point $\boldsymbol{P}(s)$, the elastic deformation of the material induces internal moments $\boldsymbol{M}^I(s)$ (departure from the rest



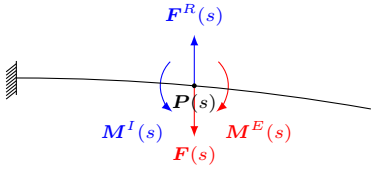
(a) Continuous representation of a beam: mean curve $\mathcal{C}$, section $\mathcal{S}(s)$ and frame $\boldsymbol{\mathcal{R}}(s)$.



(b) Reference, calculated and equilibrium configurations.



(c) External forces.



(d) Beam's reaction to the external stresses.

Fig. 2. Actors of the continuous model.

configuration). We assume here for simplicity a linear constitutive law (Hooke's law). Classical beam theory [4] allows to compute those moments (Eq. (4)), as a function of the difference between the reference and actual generalized curvatures $\underline{\boldsymbol{\Omega}}$ and $\boldsymbol{\Omega}$:

$$\boldsymbol{M}^I(s) = \boldsymbol{\mathcal{R}}(s)\mathbb{C}(s)\boldsymbol{\mathcal{R}}(s)^T\left(\underline{\boldsymbol{\Omega}}(s) - \boldsymbol{\Omega}(s)\right) = \mathbb{K}(s)\left(\underline{\boldsymbol{\Omega}}(s) - \boldsymbol{\Omega}(s)\right), \tag{4}$$

where $\mathbb{K}(s)$ is the stiffness matrix. Note that the Hooke matrix $\mathbb{C}(s)$ expressed in the local frame $\boldsymbol{\mathcal{R}}(s)$ is diagonal. Its coefficients are the twist rigidity $C_{\boldsymbol{H}}(s)$ (in the plane $(\boldsymbol{L}(s), \boldsymbol{U}(s))$, as a function of section $\mathcal{S}(s)$ and shear modulus G) and the flexural rigidities $C_{\boldsymbol{L}}(s)$ and $C_{\boldsymbol{U}}(s)$ (respectively in the planes $(\boldsymbol{U}(s), \boldsymbol{H}(s))$ and $(\boldsymbol{H}(s), \boldsymbol{L}(s))$, as a function of section $\mathcal{S}(s)$ and young modulus E):

$$\mathbb{C}(s) = \begin{bmatrix} C_{\boldsymbol{H}}(s) & \cdot & \cdot \\ \cdot & C_{\boldsymbol{L}}(s) & \cdot \\ \cdot & \cdot & C_{\boldsymbol{U}}(s) \end{bmatrix} \quad ; \quad \begin{cases} C_{\boldsymbol{H}}(s) = G\int_{\mathcal{S}(s)} u^2 + v^2 \,\mathrm{d}S \\ C_{\boldsymbol{L}}(s) = E\int_{\mathcal{S}(s)} u^2 \,\mathrm{d}S \\ C_{\boldsymbol{U}}(s) = E\int_{\mathcal{S}(s)} v^2 \,\mathrm{d}S \end{cases}, \tag{5}$$

where (u, v) are the coordinates in the plane $(\boldsymbol{L}(s), \boldsymbol{U}(s))$, with origin $\boldsymbol{P}(s)$.

When external forces $\boldsymbol{f}$ (such as the weight $\boldsymbol{f} = \rho\boldsymbol{g}$, Fig. 2c) are applied to the branch, external moments are induced. They result exclusively from the force densities $\boldsymbol{f}([s, L])$ present downstream of $\boldsymbol{P}(s)$. Denoting $\boldsymbol{F}(s) = \int_s^L \boldsymbol{f}(u)\,\mathrm{d}u$ the external force applied to segment $[s, L]$ due to gravity, we can express the

external moments as a function of forces $\boldsymbol{F}$ and tangents $\boldsymbol{H}$:

$$\boldsymbol{M}^E(s) = \int_s^L (\boldsymbol{P}(u) - \boldsymbol{P}(s)) \times \boldsymbol{f}(u)\,\mathrm{d}u = \int_s^L \boldsymbol{H}(u) \times \boldsymbol{F}(u)\,\mathrm{d}u. \tag{6}$$

At equilibrium, the internal torque (induced by deformation) exactly balances the external torque (induced by external forces) (Fig. 2d):

$$\mathbb{K}(s)\,(\underline{\boldsymbol{\Omega}}(s) - \boldsymbol{\Omega}_{eq}(s)) + \boldsymbol{M}^E(s) = \boldsymbol{0}, \tag{7}$$

where $\boldsymbol{\Omega}_{eq}$ denotes the generalized curvature at equilibrium:

$$\boldsymbol{\Omega}_{eq}(s) = \underline{\boldsymbol{\Omega}}(s) + \mathbb{K}(s)^{-1}\boldsymbol{M}^E(s). \tag{8}$$

2.2 FDM Discretization and Natural Integration Using L-Systems

Let us discretize the curve $\mathcal{C}$ into a set of $I+1$ nodes N_i of curvilinear abscissa $s_i, i = 0 \ldots I$ (usually regularly spaced though not necessarily) so that $N_0 = \boldsymbol{P}(0)$ and $N_I = \boldsymbol{P}(L)$. Each node is associated with its position $\boldsymbol{P}_i$, frame $\boldsymbol{\mathcal{R}}_i$, external moments $\boldsymbol{M}_i^E$ or accumulated downstream forces $\boldsymbol{F}_i$. If distances $\mathrm{d}s_i = \|s_{i+1} - s_i\|$ are small enough, we can express (1), (2), and (6) thanks to Taylor's series at order 1 (Euler methods) [12].

Interestingly, point $\boldsymbol{P}_{i+1}$ and frame $\boldsymbol{\mathcal{R}}_{i+1}$ can be recursively expressed in terms of the previous point $\boldsymbol{P}_i$ and frame $\boldsymbol{\mathcal{R}}_i$, which allow us to compute these quantities in a single pass from the basis of the curve to its tip [18].

$$\boldsymbol{P}_{i+1} = \boldsymbol{P}_i + \mathrm{d}s_i\,\boldsymbol{H}_i, \tag{9}$$

$$\boldsymbol{\mathcal{R}}_{i+1} = \boldsymbol{\mathcal{R}}_i + \mathrm{d}s_i\,[\boldsymbol{\Omega}_i]_\times \boldsymbol{\mathcal{R}}_i. \tag{10}$$

Likewise, external moments $\boldsymbol{M}_{i-1}^E$ and accumulated forces $\boldsymbol{F}_{i-1}$ can be recursively expressed in terms of $\boldsymbol{M}_i^E$ and $\boldsymbol{F}_i$ at the next node. Their computation can thus be carried out in a single pass from curve tip to basis.

$$\boldsymbol{M}_{i-1}^E = \boldsymbol{M}_i^E + \mathrm{d}s_{i-1}\,\boldsymbol{H}_i \times \boldsymbol{F}_i, \tag{11}$$

$$\boldsymbol{F}_{i-1} = \boldsymbol{F}_i + \int_{s_{i-1}}^{s_i} \boldsymbol{f}(u)\,\mathrm{d}u. \tag{12}$$

Due to large deformations, (7) is non-linear in terms of generalized curvature. To solve it, we use an explicit iterative method, and, specifically, a relaxation method [12] with a factor $r \in]0, 1[$:

$$\boldsymbol{\Omega}^{t+1}(s) = (1-r)\boldsymbol{\Omega}^t(s) + r(\underline{\boldsymbol{\Omega}}(s) + \mathbb{K}^t(s)^{-1}\boldsymbol{M}^{E^t}(s)), \tag{13}$$

with $\boldsymbol{\Omega}^0(s) = \underline{\boldsymbol{\Omega}}(s)$. The iterative process stops when the difference between two successive solutions is smaller than a tolerance $\|\boldsymbol{\Omega}^{t+1} - \boldsymbol{\Omega}^t\| < \varepsilon$.

The above recursive formulation makes it possible to define local L-system rules that will propagate in two pass across the branch structure, from node to node. The flow of computation goes as follows between two time steps:

```
Input: branch at time t
Output: branch at time t+1
do:
  L-system pass from tip to basis
    # computation of (11), (12), (13)
  L-system pass from basis to tip
    # computation of (9), (10)
until convergence condition of (13) reached
```

Sketch of a L-system rule used for the tip-to-basis pass

```
N --> { ds = abs( successor().s - current().s )
        current().F = successor().F  + ds * successor().f
          # ... computation of (11) and (13)
      } produce N
```

Sketch of a L-system rule used for the basis-to-tip pass

```
N --> { ds = abs( predecessor().s - current().s )
        current().P = predecessor().P + ds * predecessor.H
          # ... computation of (9), (10)
      } produce N
```

3 Natural Computing of Branch Bending Using Finite Element Method (FEM) and L-Systems

3.1 Computing Axis Bending by Axial Information Propagation with FEM

In FDM and FEM, continuous model domains are approximated using information at a finite number of discrete locations called nodes $N_i, i = 0, \ldots, I$. Whereas in FDM, solutions are only evaluated at nodes (and not elsewhere within the domain), in FEM the set of nodes correspond to the vertices of polygonal elements that tile the domain of interest. The solution is evaluated at each node using an integral formulation and interpolated over the whole domain using a basis of *shape functions* φ_i associated with each node N_i) [2]. Here, our aim is to compute the generalized curvature $\boldsymbol{\Omega}$ that characterizes the axis shape on the whole domain (i.e. on the curve $\mathcal{C}$). For this we decompose $\boldsymbol{\Omega}$ on the set of shape functions:

$$\boldsymbol{\Omega}(s) = \sum_{i=0}^{I} \boldsymbol{\Omega}_i \varphi_i(s), \tag{14}$$

where $\boldsymbol{\Omega}_i$ is a vector. Shape functions φ_i are usually low order polynomials that are null on all node $N_j \neq N_i$ and have value 1 at node N_i. They are interpolating and form a partition of unity [2]. Their support is compact and their values at one node influences those of neighboring elements.

To compute values $\boldsymbol{\Omega}_i$ on nodes N_i, we have to solve the linear system $\mathbb{M}\boldsymbol{X} = \boldsymbol{B}$ defined by 15, [2]:

$$\sum_{i=0}^{I} \underbrace{\boldsymbol{\Omega}_i}_{=\boldsymbol{X}_i^T} \underbrace{\int_{\mathcal{C}} \varphi_i(s)\varphi_j(s)\,\mathrm{d}s}_{=\mathbb{M}_{ji}} = \underbrace{\int_{\mathcal{C}} \underline{\boldsymbol{\Omega}}(s)\varphi_j(s)\,\mathrm{d}s + \int_{\mathcal{C}} \mathbb{K}(s)^{-1}\boldsymbol{M}^E(s)\varphi_j(s)\,\mathrm{d}s}_{=\boldsymbol{B}_j^T}, \quad (15)$$

where $\mathbb{M}_{ij}$ correspond to the energy of the cross influence of nodes N_j and N_i on the axis, $\boldsymbol{X}_i = \boldsymbol{\Omega}_i^T$ and $\boldsymbol{B}_i$ to the energy of forces along the axis which influence the generalized curvature $\boldsymbol{\Omega}_i$ of the node N_i. If the mass-matrix coefficient $\mathbb{M}_{ji}$ can be analytically computed (shape function are known) and expressed as a sum of integrals on each element, we have to compute numerically the right hand-side $\boldsymbol{B}_j$. Because this term is not linear, we split up each element in several integration areas and use midpoint method [12] to numericaly approach the integrals (note that one may also use the Gauss points method [12]).

Properties of mass-matrix (symmetric and positive definite) allow us to use a Cholesky decomposition [12] (product of a low triangular matrix with its transpose) to solve in two data propagation through the structure thanks to forward substitution (17) and backward substitution (18) algorithms [12].

$$\mathbb{M} = \mathbb{L}\mathbb{L}^T \quad , \quad \begin{cases} \mathbb{L}_{ij} = \dfrac{\mathbb{M}_{ij} - \sum_{k=0}^{j-1} \mathbb{L}_{ik}\mathbb{L}_{jk}}{\mathbb{L}_{jj}}, \forall 0 \leqslant j < i \leqslant I \\ \mathbb{L}_{ii} = \sqrt{\mathbb{M}_{ii} - \sum_{k=0}^{i-1} \mathbb{L}_{ik}^2}, \quad \forall 0 \leqslant i \leqslant I \end{cases} \quad (16)$$

$$\mathbb{L}\boldsymbol{Y} = \boldsymbol{B} \quad , \quad \boldsymbol{Y}_i = \frac{\boldsymbol{B}_i - \sum_{k=0}^{i-1} \mathbb{L}_{ik}\boldsymbol{Y}_k}{\mathbb{L}_{ii}}, \forall 0 \leqslant i \leqslant I \quad (17)$$

$$\mathbb{L}^T\boldsymbol{X} = \boldsymbol{Y} \quad , \quad \boldsymbol{X}_i = \frac{\boldsymbol{Y}_i - \sum_{k=i+1}^{I} \mathbb{L}_{ki}\boldsymbol{Y}_k}{\mathbb{L}_{ii}}, \forall I \geqslant i \geqslant 0 \quad (18)$$

Cholesky decomposition (16) and forward substitution (17) algorithms can be computed together with one pass, e.g. from basis-to-tip (resp. from tip to basis) and the backward substitution (18) algorithm can be computed with an a pass in the reverse direction, e.g. from tip to basis (resp. from basis to tip).

3.2 Extension to Branching Systems

We now need to extend the previous algorithm so that it can cope with branching organizations of beams that would represent plant structures. As in a branching structure, each element has only one parent, ramifications do not influence forward propagations (update of frames $\boldsymbol{\mathcal{R}}(s)$ and points $\boldsymbol{P}(s)$).

Solving the linear system $\mathbb{M}\boldsymbol{X} = \boldsymbol{B}$ is more difficult in case of ramification than in the case of a single axis. Non-null elements $\mathbb{M}_{ij}$ in the matrix $\mathbb{M}$ correspond to branch segments between nodes N_i and N_j such that the product of the shape functions φ_i and φ_j along these segments is non-null. Therefore, the position of non-null elements in $\mathbb{M}$ depends on the indexing of the tree nodes. We

consider two indexing strategies: a forward and a backward strategies indexing respectively the elements from basis to tip (matrix $\mathbb{M}^f$) and from tip to basis (matrix $\mathbb{M}^b$). Using either of indexing strategies, matrices have a block structure according to the set of nodes between two branching points (Fig. 3).

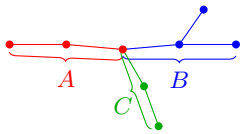
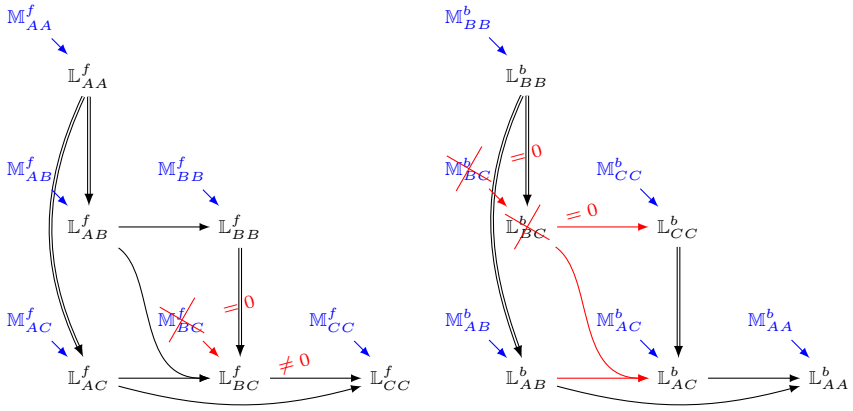


Fig. 3. Sets of nodes corresponding to each block of matrices $\mathbb{M}^f$ and $\mathbb{M}^b$.

$$\mathbb{M}^f = \begin{bmatrix} \mathbb{M}^f_{AA} & & sym \\ \mathbb{M}^f_{AB} & \mathbb{M}^f_{BB} & \\ \mathbb{M}^f_{AC} & \cdot & \mathbb{M}^f_{CC} \end{bmatrix} \quad ; \quad \mathbb{M}^b = \begin{bmatrix} \mathbb{M}^b_{BB} & & sym \\ \cdot & \mathbb{M}^b_{CC} & \\ \mathbb{M}^b_{AB} & \mathbb{M}^b_{AC} & \mathbb{M}^b_{AA} \end{bmatrix}. \tag{19}$$

With the same notations, we can compute $\mathbb{L}^f$ and $\mathbb{L}^b$ the Cholesky decomposition matrices of $\mathbb{M}^f = \mathbb{L}^f {\mathbb{L}^f}^T$ and $\mathbb{M}^b = \mathbb{L}^b {\mathbb{L}^b}^T$ respectively. Then, building the directed acyclic graphs that correspond to data propagation in Cholesky decomposition algorithm. It is possible to show that only the Cholesky decomposition $\mathbb{L}^b$ keeps non-null coefficients at exactly the same places as those of the original matrix $\mathbb{M}^b$ (Fig. 4) [6].



(a) Forward indexing. (b) Backward indexing.

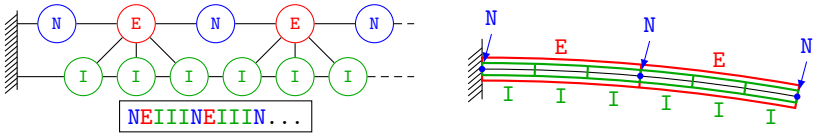
Fig. 4. Direted acyclic graphs that correspond to data propagation in Cholesky decomposition algorithm. With a forward indexing, $\mathbb{L}^f_{BC} \neq 0$ whereas $\mathbb{M}^f_{BC} = 0$ contrary to a backward indexing where $\mathbb{L}^b_{BC} = 0 = \mathbb{M}^b_{BC}$.

3.3 Natural Computing Using L-System

On an axis, elements and integration domains are segments. Since a node has influence only on its neighboring elements (possibly at order greater than 1), we can express our model in L-systems:

- a node is represented by a module of type N,
- an element between two nodes is represented by a module of type E,
- elements E are decomposed into integration segments represented by modules of type I.

Because two elements can be decomposed into two different number of integration segments, and a node influences always the same number of neighboring elements, we chose to use a multiscale L-string representation [3] to carry out the integral calculus. Thus the axis is represented at two scales: the scale of nodes and elements and the scale of integration points. The first scale is used to assemble the mass-matrix $\mathbb{M}^b$ and solve the linear system $\mathbb{M}^b\boldsymbol{X} = \boldsymbol{B}$ whereas the second scale is used to compute $\boldsymbol{B}$.



(a) Axis seen as multiscale L-string for FEM model (L-string and tree graph).

(b) Each L-string module corresponds to a part of the axis.

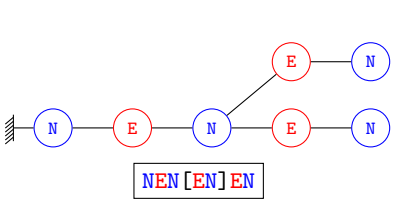
Fig. 5. Different representations of a multiscale L-string.

stored in the node N_i.

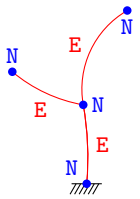
When a ramification exists, we deal with it in L-system by adding brackets after a node N to begin a new axis having this node as a root. The L-string NEN[EN]EN corresponds to a simple branch composed of a segment axis E divided in two axis segments (Fig. 6a and 6b). Like previously, each element E is decomposed into several integration segments I at a lower scale.

Using this data structure and storing each row of matrices from their diagonal to their last coefficient in the corresponding node, it is possible to compute the Cholesky decomposition and the forward substitution (and therefore all the mechanical quantities) in a tip-to-basis pass using the following algorithm:

```
Input: 𝕄, B and order of shape functions n
Output: 𝕃 and Y

init:
  N --> { current().𝕃^tmp = current().𝕄
          current().Y^tmp = current().B
        } produce N
```



(a) L-string at elements and nodes scale for FEM model in case of ramification (L-string and tree graph).



(b) Each L-string module is equivalent to a part of the tree at elements and nodes scale.

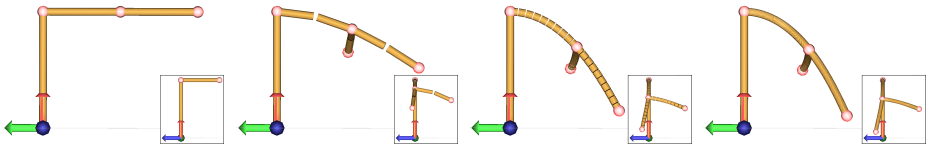
Fig. 6. Different representations of a ramification L-string at nodes and elements scale.

```
# Cholesky decomposition:
  N --> { current().𝕃₀ = √(current().𝕃₀^tmp)
          for i = 1...n:
            current().𝕃_i = current().𝕃_i^tmp / current().𝕃₀
            forall k, p in { predecessors() of order k ⩽ i }:
              p.𝕃_i^tmp = p.𝕃_i^tmp − current().𝕃_k * current().𝕃_i

        } produce N

# Forward substitution:
  N -->  { current().Y = current().Y^tmp / current().𝕃₀
           forall i, p in { predecessors() of order i ⩽ n }:
             p.Y^tmp = p.Y^tmp - current().𝕃_i * current().Y

         } produce N
```

4 Results

We first tested our algorithm on a simple branching system composed of a rigid trunk, a horizontal branch and a secondary branch borne by the former one. The method is able to account for bending and twist, Fig. 7. Only few nodes were needed (here, only at each end of the branch and at each of its ramification nodes) to obtain curvature along the axis (Fig. 7d). Note that if we do not have enough integration points (Fig. 7b), the number of nodes and integration points are not enough to converge correctly.

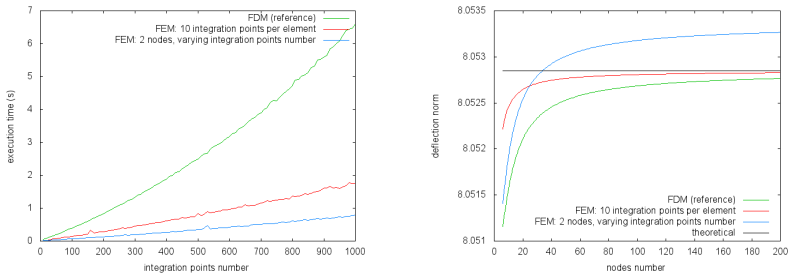
To analyze this resolution issue, we compared our result to the model presented in the section 2 (green curves in Fig. 8). We present two simulations:



(a) Reference configuration. (b) 2 points per element. (c) 10 points per element. (d) 100 points per element.

Fig. 7. Branch bending with one ramification. 1 node (red spheres) at each end and ramification. Integration points are located on the midpoint of each brown segment (integration areas).

- one with only two nodes (at the beginning and at the end of the axis): we are only varying the number of integration points (blue curves in Fig. 8),
- another one where we are varying the number of nodes and where the number of integration points per element is fixed to 10 (red curves in Fig. 8).



(a) Execution time (in seconds) as function of integration points number (in FDM, integration points and nodes numbers are the same).

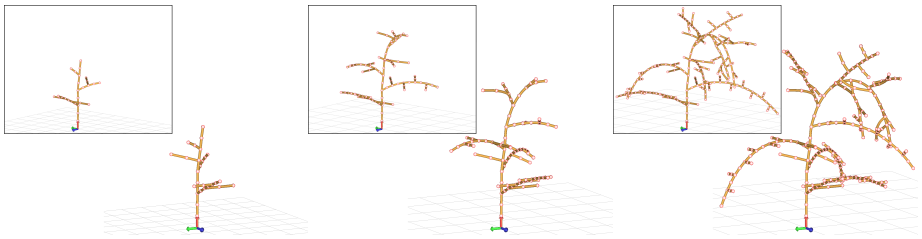
(b) Convergence (norm of the deflection) as function of nodes number except for blue curve: as function of integration points number.

Fig. 8. Performances of our method on a single axis bending compared to FDM (reference mode, green curves). Two approaches are studied: nodes number fixed and increase the points integration numbers (2 nodes, blue curves) ; increase nodes number with fixed integration points number per element (red curves).

Fig. 8a shows us that our method is faster than a finite difference method. In general, the execution time increases roughly linearly with the number of integration points. Furthermore, for a given number of integration points, the less nodes we use the faster is our method.On Fig. 8b, we observe that our method converges more rapidly than a FDM method for a similar number of nodes. The error (distance between the simulated and the theoretical values) is a decreasing function of number of nodes. However, decreasing the number of integration points does not change the convergence speed but may affect the

convergence itself (blue curve). A minimal density of integration points must therefore be used to obtain correct physical results.

Our method allows to compute branch bending with different kinds of growth rules (Fig. 9): we can play with reference curvature, material properties (density, Young and shear modulii, ...), order of ramifications, children number at each ramification, sections, segment's length...



(a) 3 growth steps. (b) 6 growth steps. (c) 10 growth steps.

Fig. 9. Branch bending on growing tree with 2 perspectives.

5 Conclusion

In this paper, we extended FDM to FEM integration in L-systems. For this we had to use a multiscale approach where the plant is represented at two scales to model both the nodes and the integration points of a FEM approach. We showed that we could solve symmetric and definite positive linear systems thanks to a Cholesky decomposition in L-systems, that made it possible to use the branching structure itself to propagate the numerical integration as a flow of information from the basis of the plant to the tip and reciprocally.

Our comparative analysis showed that our L-system FEM converges more rapidly for our application than L-system FDM (with same model). This approach, illustrated on a mechanical problem of branch bending, can be readily extended to the resolution of other systems involving differential equations on branching systems.

References

1. Allen, M. and Prusinkiewicz, P. and DeJong, T. M.: Using L-systems for modeling source-sink interactions, architecture and physiology of growing trees: the L-PEACH Model. New Phyotologist 166, 869–880 (2005)
2. Bathe, K.: Finite Element Procedures. Prentice Hall (1996)
3. Boudon, F. and Pradal, C. and Cokelaer, T. and Prusinkiewicz, P., and Godin, C.: L-Py: an L-system simulation framework for modeling plant architecture development base on a dynamic language. Frontiers in Plant Science 3(76) (2012)

4. Chou, P. C. and Pagano, N. J.: Elasticity: tensor, dyadic, and engineering approaches. Courier Dover Publications (1992)
5. Costes, E. and Smith, C. and Renton, M. and Guédon, Y. and Prusinkiewicz, P. and Godin, C.: MAppleT: simulation of apple tree development using mixed stochastic and biomechanical models. Functional Plant Biology 35(10) (2008)
6. Featherstone, R.: Efficient Factorization of the Joint-Space Inertia Matrix for Branched Kinematic Trees. The International Journal of Robotics Research 24(6), 487–500 (2005)
7. Federl, P. and Prusinkiewicz, P.: Solving differential equations in developmental models of multicellular structures expressed using L-systems. Computational Science – ICCS 2004 pp. 65–72 (2004)
8. Godin, C. and Sinoquet, H.: Functional-structural plant modelling. The New Phytologist 166(3), 705–708 (2005)
9. Hemmerling, R. and Evers, J. B. and Smoleov, K. and Buck-Sorlin, G. and Kurth, W.: Extension of the GroIMP modelling platform to allow easy specification of differential equations describing biological processes within plant models. Computers and Electronics in Agriculture 92(C), 1–8 (2013)
10. Jirasek, C. and Prusinkiewicz, P. and Moulia, B.: Integrating biomechanics into developmental plant models expressed using L-systems. Plant Biomechanics 24(9), 614–624 (2000)
11. Peiró, J. and Sherwin, S.: Finite Difference, Finite Element and Finite Volume Methods for Partial Differential Equations. Dordrecht: Springer Netherlands Handbook of Materials Modeling, 2415–2446 (2005)
12. Press, W. H., and Teukolsky, S. A. and Vettering, W. T. and Flannery, B. P.: Numerical Recipes: The art of scientific computing. Cambridge University Press (1987)
13. Prusinkiewicz, P.: Geometric modeling without coordinates and indices. IEEE Computer society Proceedings of the IEEE Shape Modeling International., 3–4 (2002)
14. Prusinkiewicz, P.: Modeling plant growth and development. Current Opinion in Plant Biology 7(1), 79–83 (2004)
15. Prusinkiewicz, P. and Allen, M. and Escobar-Gutierrez, A. and DeJong, T. M.: Numerical methods for transport-resistance sink-source allocation models. Frontis 22, 123–137 (2007)
16. Prusinkiewicz, P. and Lindenmayer, A.: The algorithmic beauty of plants. Springer (1990)
17. Prusinkiewicz, P. and Runions, A.: Computational models of plant development and form. The New Phytologist 193(3), 549–569 (2012)
18. Taylor-Hell, J.: Incorporating biomechanics into architectural tree models. Computer Graphics and Image Processing SIBGRAPI 2005. 18th Brazilian Symposium on. IEEE (2005)
19. Vos, J. and Evers, J. B. and Buck-Sorlin, G. H. and Andrieu, B. and Chelle, M. and de Visser, P. H. B.: Functional-structural plant modelling: a new versatile tool in crop science. Journal of Experimental Botany 61(8), 2101–2115 (2010)