THE UNIVERSITY OF CALGARY

Biomechanics in Botanical Trees

by

Julia Taylor-Hell

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

September, 2005

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled “Biomechanics in Botanical Trees” submitted by Julia Taylor-Hell in partial fulfillment of the requirements for the degree of Master of Science.

Supervisor, Dr. Przemyslaw Prusinkiewicz
Department of Computer Science

Dr. Lawrence D. Harder
Department of Biological Sciences

Dr. Faramarz Samavati
Department of Computer Science

Date

Abstract

In this thesis, I describe a method for creating tree models with realistically curved branches, by considering the tree's development in the context of its environment. The final shape of the branches results from their growth under the influence of gravity and tropisms. Using the framework of L-systems, I extend Jirasek's biomechanical simulation of a plant axis to correctly represent entire trees. I present simulations of *reaction wood* and branch breakage due to excessive stress, as well as methods for controlling tree architecture. The result of all these factors is a multi-year biomechanical simulation of tree growth, which produces a realistic tree shape at every stage of its development. As a case study, my model is used to represent the mutant *crooked poplar* tree. I also investigate the dynamics problem of representing oscillatory motion of tree branches. I solve a two-dimensional problem using non-inertial systems to model a branched system of spring-linked segments. The remainder of the dynamics solution is left as future work.

Acknowledgements

I would like to express my gratitude to my supervisor Dr. Przemyslaw Prusinkiewicz for his guidance in the completion of this thesis, and for his infectious enthusiasm. I am also grateful Dr. William Remphrey for his botanical insights and communications about the crooked poplar tree, as well as Prof. Alfredo Louro for his generous assistance with the dynamics problem. I would like to thank Mark Myhre for his encouragement, as well as my father Pavol, mother Susan and sister Catherine for their support and for reading through endless chapters. I am grateful for the help of Mikolaj Cieslak, Peter MacMurchy, Kevin Foster and Kelly Poon. I very much appreciate the financial support provided for this work by the Natural Sciences and Engineering Research Council of Canada and the Alberta Informatics Circle of Research Excellence.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Botanical trees are modelled extensively for the portrayal of natural scenes in the animation industry. Obtaining realistic models of trees is a common goal in this field as well as in botanical research, horticulture and forestry [12]. Instead of painstakingly reproducing observed characteristics or using data capture from real trees, the goal of this thesis is to generate realistic models from the results of a biomechanical simulation. With this approach, tree shapes and behaviours can be obtained as they are in nature – as the result of their physical makeup in interaction with their environment.

An important shape characteristic of many trees seen in nature is the bending and curving of branches. These shapes result from the influences of gravity and tropisms while a branch is undergoing radial growth [15]. Jirasek used L-systems to express a biomechanical model of a growing plant axis that responds to these influences [25]. One contribution of this thesis is to extend her method to handle the biomechanics at branching points, allowing complete representations of trees. By using a new capacity for fast information transfer in L-systems, the goal is to significantly speed up simulations.

Another contribution of this thesis involves the simulation of wood properties, based in a study of elasticity theory. In nature, reaction wood is produced to actively reorient a leaning branch, resulting in a curved trunk which regains its desired direction of growth. To accomplish similar results in the model, I simulate several aspects of reaction wood production described in botanical literature. Breakage of branches due to excessive stresses from weight, as experienced by some fruit trees, is also simulated.

To visualize the influence of biomechanics on tree growth, one must begin with a realistic tree architecture. In this thesis, two methods of controlling tree architecture are presented.

The first method is descriptive; it specifies stochastic rules for tree growth based on observations of natural trees. Such rules are incorporated into the model by means of graphical functions which can be edited by the user. The second method allows tree architecture control to emerge from a physiologically-based simulation of resource allocation, symbolically based on the flow of water and minerals throughout a growing tree.

I also investigate the dynamics problem of representing oscillatory motion of tree branches in response to external forces, such as the quick pulling of a branch. With such a representation, realistic motion could be generated automatically, as a result of a physically-based simulation, instead of being created by animators. I solve a two-dimensional problem using non-inertial systems to model a small branched system of spring-linked segments under the influence of gravity. The remainder of the dynamics solution is left to future work.

The thesis is organized as follows. Relevant previous work is discussed in Chapter 2. Chapter 3 is a description of L-systems, the framework used to express the tree models. In Chapter 4, two approaches to controlling tree architecture are presented, and the tree's seasonal cycle of development is described. An introduction to the theory of elasticity as it affects the models is given in Chapter 5. In Chapter 6, an in-depth description of the biomechanical simulation is given, reviewing Jirasek's method for single plant axes and extending her model to branching structures. Chapter 7 is a case study of the "crooked" mutant of the *Populus tremuloides* species, a tree composed of curved and twisted branches caused by severe bending due to gravity. The simulations of reaction-wood production and breakage are described in Chapter 8. Chapter 9 provides an investigation of the dynamics problem, using non-inertial systems. Conclusions and directions for future work are discussed in Chapter 10. The most important L-system code for the models described in this thesis is presented in the Appendix.

Chapter 2

Previous Work

Much research has gone into the development of realistic tree models, because of its importance in both the computer graphics industry and the biological modelling community. General methods of modelling trees for computer graphics have been reviewed in [40] and [12]. As the field of tree modelling is very broad, this chapter focuses on reviewing works related to biomechanics. Background for other components of the models will be introduced in the relevant chapters.

Several methods have been used to create physically-based tree models to represent branch bending. Though the approaches range from trajectories of particle systems [48] to fluid flow [49], most of the work has focused on either deformable beams or spring-linked segments. A survey of physically-based tree modelling techniques for semi-static simulations is presented in Section 2.1. In particular, previous work is described which represents bending of branches due to weight and tropisms. In Section 2.2, previous work in representing the dynamic motion of tree branches is described. In Section 2.3, I outline some relevant work from the fields of hair modeling and animation. Section 2.4 is a review of some botanical research relevant to the model.

2.1 Modelling for statics

Statics is the study of objects at rest [19]. Static tree modelling methods balance internal and external forces to achieve an equilibrium position, or shape, for the tree. These methods can be used alongside a growth simulation in order to represent a tree's equilibrium shape at each step during its development.

Hart et al. [21] have created such a developmental tree model in which growth is partly

controlled by the solution to a static system. In their model, *growth rates* of branches were computed based on a compromise between two goals – balancing weight around the centre of mass and maximizing the results of photosynthesis. Torques acting on the trunk were reduced by balancing the tree's centre of mass over its trunk. The growth of branches was accelerated or inhibited in response to a centre-of-mass calculation. However, branches in their model did not curve or alter their orientation in response to gravity.

Other authors have sought to simulate the curving of branches due to weight and tropisms. Archer [5] described the curvature of a tree stem as being affected by three phenomena:

1. bending from self-weight (long-term loading),
2. stem rigidification due to radial growth, and
3. secondary reorientation caused by differentiation of material within a branch.

Fournier et al. [15] presented a biomechanical model using beam theory to approximate the bending of a plant stem due to loading and stresses. They performed a biomechanical study of tree growth, which included the development a mechanical model for stresses and strains in a cross-section. Their model described the effect of added radial layers on these stresses and on stem shape. As a result, the authors predicted a tree stem characterized by an "S" shape visible in many plant axes.

More recent advances have made use of these results. Alméras et al. [3] modelled a two-dimensional branch using a chain of beams that bend due to masses concentrated at their base. In their model, bending moments and beam curvatures were continually recalculated until equilibrium was reached. Fourcaud et al. [13] used finite-element analysis to represent the mechanical behaviour of growing trees. Their model was composed of a sequence of multi-layer deformable beam segments which grew radial layers to simulate thickening of the branch. New beams were added to simulate branch lengthening.

Instead of representing a tree stem as a continuously bending beam, the structure may be

discretized and the bending of a branch modelled by rotating a set of joints between small straight segments. This representation is similar in structure to the *mechanical manipulator* robot described in detail by Craig [10]. The most intuitive way to specify orientations and positions of the segments is relationally, describing each in terms of its deviation from the previous segment. With this approach, the orientation of any segment implicitly accounts for rotations of ealier segments.

Jirasek [25] employed L-systems to create a biomechanical model of a growing plant axis that bends due to external forces. In her model, physical properties were propagated in both directions along a growing axis, resulting in the shape of the axis at static equilibrium. Jirasek's model dealt largely with single plant axes. Although branching structures were represented, the biomechanical simulation at branching points was not handled because her approach relied on the assumption of infinitesimal rotations. Her work was also limited by the complexity of the model's representation in CPFG, the modelling framework for L-systems in use at the time.

Recent development of L-system languages has enabled extension of Jirasek's approach. Karwowski introduced the L+C modelling language [29], which makes two important improvements to the workings of L-systems relevant to modelling biomechanics. First, it allows user-defined structures to be passed as parameters to L-system modules. Thus, multiple parameters such as torques and velocities may be carried together in a single structure. Previously, only simple types were allowed as parameters, which made the lists of production parameters prohibitively complex for physically-based models such as Jirasek's.

Also, L+C introduces *fast information transfer*, described in Chapter 3, by which a signal originating at one end of a structure can be propagated to the other end in a single rewriting step. This speeds up simulations significantly, allowing for interactive frame rates and much more complex models.

Part of this thesis is focused on extending Jirasek's method to handle the biomechanics

of branching structures, so that entire trees may be simulated.

2.2 Modelling for dynamics

Several researchers have proposed systems to model the dynamics of trees. Often, the goal has been to represent wind acting on tree branches, causing them to sway.

Shinya and Fournier [48] created a simulation of the dynamics of tree branches in wind. The authors used a combination of stochastic and physically-based methods to reduce the required computation. Their simulation of the wind and dynamic models were kept separate from the deformation models, so that different geometric descriptions of the tree model could be used with the same system. The authors approximated dynamic motion of various types of tree models either by using damped coupled harmonic oscillators or by synthesizing motion from the vibration modes of a uniform beam.

Stam [50] built on this work, creating efficient dynamics simulations by modelling the effect, instead of the cause, of the movement. He used modal analysis to create a filter of noise in the Fourier domain. Good results were obtained by superimposing precomputed characteristic shapes for a small number of modes to obtain the resulting shape for any particular force. With this method, he eliminated the need to integrate dynamics equations, thereby obtaining real-time results.

Several other dynamics approaches use a two-pass method which propagates physical information forward and then backward along a branch. Jones et al. [26] used a simple L-system tree model to represent deflection, resistance and twist of branches due to simulated wind. The techniques used were described as *pseudophysics*, as their goal was to produce visually pleasing results quickly, instead of achieving physical accuracy.

In the work of Sakaguchi et al. [46], external forces acting on each segment were calculated in a forward pass from the branch base to the tip, and then restoration forces were calculated in the reverse direction from tip to base. Finally, the rotations were applied to each

segment and they were reconnected and oriented so that the rotations calculated became relative to the end of the previous segment. This last step is less elegant than it could be if the author had used a relational representation and non-inertial reference frames; segments that remain attached and rotate in relation to each other are the natural product of such a framework.

Armstrong developed the theory for simulating dynamics of tree branches in wind using non-inertial reference frames and the associated non-inertial forces. This work also used bidirectional information propagation, collecting physical information in separate forward and backward passes. The author made some simplifying assumptions to reduce the required computation, claiming that link rotations could be considered infinitesimal and as such could be represented as vectors. He also ignored these rotations during some intermediate calculations. These simplifications limited the model to representing trees swaying in a slight breeze, as the deviation from the initial state must be relatively small at any stage.

In the case of wind, it is difficult to discern whether the visual results are physically accurate, as one cannot see the wind which is causing the motion. Thus, even ad hoc methods can produce convincing results. Creation of believable motion in response to visible stimuli, such as a hand quickly pulling the end of a branch, is more difficult because the viewer is cued as to what motion to expect.

2.3 Similarity to hair modelling and animation

A study of the static and dynamic behaviour of tree branches has much in common with the fields of hair modelling and animation. As such, techniques suggested in this field are applicable to the problem of tree modelling. Explicit models which represent the dynamics of individual hair strands are particularly relevant to the simulation of plant axes.

Daldegan et al. [11] used a mass-spring-hinge model, similar to the spring-linked segment model for tree modelling, to represent hair strands. A similar structure is also used by

Rosenblum et al. [45]. Anjyo et al. [4] also used a simplified cantilever beam divided into segments to represent a hair strand, and used this model to approximate the dynamics of hair motion. Hadap and Magnenat-Thalmann [18] conceptualized an individual hair as a string of particles connected by bending and torsional springs, ignoring the rigid tensile springs which would lead to stiff equations. As such their final representation used links attached by spherical joints with a set of reference frames that moved with the links. Quaternions were used to represent each joint's rotation. The influences of gravity and of interactions with air, body or other hairs are applied to this structure.

2.4 Botanical background

For centuries, observers of nature have formed theories about the structure of trees [44] [34] [24]. This section presents background findings and definitions relating to tree growth and structure.

The *order* of a branch refers to the number of ancestors it has in the tree; a main trunk has order 0. A branch of order j is called the *child* of the $(j-1)$-order branch from which it originated. In a growing tree, *primary growth* refers to the lengthening of branches by the growth of new *internodes* or small segments, as well as the production of *nodes*, including leaves and buds. *Secondary growth* is radial thickening that occurs to support the added primary growth [53].

2.4.1 Branch girth

With regard to the girth of branches, Da Vinci hypothesized that the cross-sectional area of a parent branch equals the sum of cross-sectional areas of its child branches [44]. In the case of *bifurcation* (branching into two lateral segments) his hypothesis translates to

$$r_0^2 = r_1^2 + r_2^2. \tag{2.1}$$

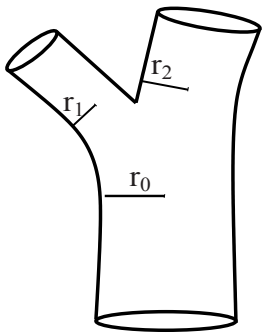


Figure 2.1: Radii r_1, r_2 of lateral branches lead to parent radius r_0

In 1927, Murray [34] performed measurements on many species of trees and came to a result which is a variant of Da Vinci's equation:

$$r_0^P = r_1^P + r_2^P \tag{2.2}$$

where P ranged between 2.49 for large trees and 3.0 for smaller trees. Murray's findings were reinforced by theoretical reasoning in 1964 with the introduction of the *pipe model* of Shinozaki et al. This theory determines branch girth by requiring that every leaf have a pipe to conduct water to it from the base of the tree [47]. Therefore the cross-section at any point along a branch must be large enough to contain pipes for all the leaves above this point in the branch. The pipe model theory has been used extensively as the basis for determining the girths of branches as a function of the number of leaves or child branches they support.

Holton [23] introduced a *strand model,* based on the pipe model. Strand elements simulate the tree's internal vasculature and are responsible for branching angles, lengths and girths of limbs. The tree model grows under the influences of gravity and light, and tropic responses are approximated.

2.4.2 Branching angles

Two changes to orientation are made at branching points. The *deviation angle* θ is the angle made by a lateral branch relative to its parent's axis, as depicted in Figure 2.2. The *divergence*

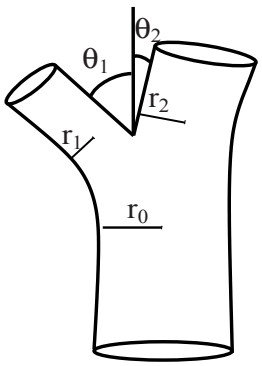


Figure 2.2: Deviation angles θ_1 and θ_2 of lateral branches

angle ϕ specifies the position around the parent axis at which lateral growth occurs. In many trees, this angle ϕ is the familiar 137.5° associated with *spiral phyllotaxis* [40]. Each new terminal bud formed along a branch is produced at a rotation of 137.5° beyond the previous bud, as in Figure 2.3.

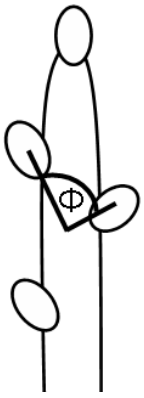


Figure 2.3: Spiral phyllotaxis in newly produced buds

Some species of trees exhibit other forms of phyllotaxis. For example, decussate phyllotaxis produces pairs of opposite buds with a divergence angle of $\phi = 90^\circ$ between pairs.

Much work has been done by Honda et al. [24] to create rules for branching in botanical tree models based on measurements of real trees. These rules specify qualities such as distance between branching points, number of new branches at each point, and branching angles in relation to the parent branch and to gravity.

2.4.3 Fate of buds

Botanists refer to a section of branch grown during a single year as the *year's growth increment*, shown in Figure 2.4. Within this increment, many buds are formed. The most vigourous buds will become new shoots during the next year, and the rest will abort or remain dormant [53].

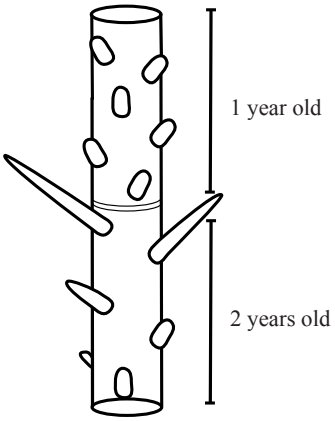


Figure 2.4: Yearly growth increment

Wilson [53] noted that, in temperate climates, the probability of a bud becoming a shoot is higher for buds produced later during the year. As a result, the largest branches occur near the top of each growth increment. He also noted that higher-order branches elongate more slowly than the main trunk [53]. In general, the length of a shoot depends on the *vigour*, or productivity, of its parent branch.

2.4.4 Tropisms

A *tropism* is a preferred direction of growth in a plant in response to environmental stimuli [53]. *Geotropism* refers to oriented growth in reaction to gravity. In particular, *negative geotropism* indicates the tendency of a new shoot to orient itself upwards, against the pull of gravity.

2.4.5 Reaction wood

Reaction wood is a modified form of wood produced by leaning branches so as to regain their desired direction of growth. Hart et al. investigated two aspects of reaction-wood modelling. First, in a simulation of the statics of tree growth, they modelled uniformly increased girth at branching points where reaction wood should be formed to resist a rotational force [21]. As the authors noted, this uniform increase in girth is only a rough approximation of reaction wood, because the cross-section of a branch producing reaction wood is not circular. In another paper, the same authors modelled the shape of branching points with reaction wood using implicit surfaces [20]. In this model, the branch cross-section is non-circular, and the different location of reaction wood in broad-leafed trees and conifers is accounted for. However, their careful modelling of reaction wood at branching points was not used in the context of a physical simulation.

Chapter 3

L-systems

L-systems are well-suited to express the problem of biomechanical plant growth. Introduced by Aristid Lindenmeyer, L-systems have been used extensively to model growing structures such as plants [40]. An L-system represents a structure as a string of *modules*, or symbols, which represent its component parts. At every derivation or *rewriting step*, each module is transformed according to a matching *production rule*. This process can be used to expand the string of modules, resulting in the growth of the model when the modules are interpreted visually.

Stochastic L-systems introduce randomness into the process of derivation, so that a module may be rewritten differently each time the L-system simulation is performed. The variation in results produced thereby allows many different individuals to be generated from the same L-system model. These individuals will have different appearance but be based on similar developmental rules, simulating individuals of the same species.

In *Parametric L-systems*, each module M can contain parameters p_i which store additional information about this part of the structure. A module with three parameters would be expressed as $M(\ p_1,\ p_2,\ p_3\ )$. Originally, only real-valued parameters were allowed [40].

3.1 Context-sensitive L-systems

Context-sensitive L-systems make use of these parameters p_i to communicate information to neighbouring modules in the string. This context information can be used within a production rule to affect the results of the rewriting step, as in Example 1. Context-sensitive L-systems are often used to simulate interaction between neighbouring modules, or to propagate a signal throughout a plant [40].

3.1.1 Example 1

The following production rule shows growth at the apex of a plant. The module *I* represents an internode and *A* represents the apex. The symbol < indicates that the parameters of the module to the left are being examined in rewriting the module to the right; in other words that *left context* is being used.

```
Axiom: I(1) A(1);


production: I(pl) < A(pa) : {
    produce I(pl + 1) A(pa + 1);
}
```

Only the module after the '<' symbol (the *strict predecessor*) is rewritten by this production; the module to the left is only used as context. Therefore, after one rewriting step, the string given in the axiom above becomes:

```
I(1) I(2) A(2).
```

Used repeatedly, this production rule will grow a stem by continually appending new internodes just before the apex. Each internode's parameter will describe its position in the sequence of internodes.

3.1.2 Example 2

The production rule given in Example 1 demonstrates the use of *left context*, and can be used to propagate a signal from left to right across the string of modules. The use of *right context* is indicated by the symbol '>', and is expressed as

```
production: I(p) > I(pr) : {
    produce I(pr} ;
}
```

In this case, the leftmost module will be rewritten, and the module to the right is used as context. Such a production can be used to propagate the signal `pr` from right to left along the string of modules.

Using left or right context, two directions of information transfer are achieved, as depicted in Figure 3.1.

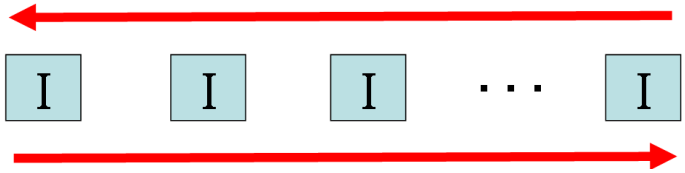


Figure 3.1: Two directions of information propagation

3.2 Branching structures

L-systems can gracefully handle the representation of branching structures. *Bracketed L-systems* use stack operations to perform calculations on one branch at a time. At a branching point, information from the parent segment's module is pushed onto the stack before computation proceeds along the first child branch. Before processing the second child branch, the parent's information is regained by popping it off the stack. As such, both children can obtain context from the parent module, as depicted in Figure 3.2.

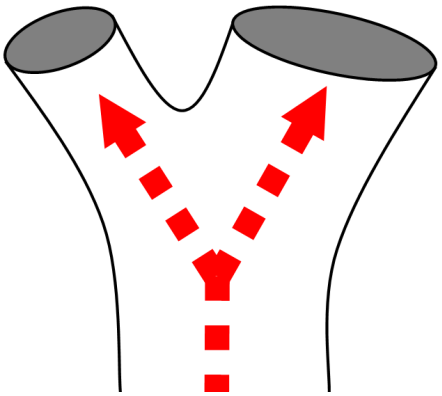


Figure 3.2: At a branching point, both children obtain context from the parent module.

3.3 Improvements in LPFG

The new L-system language *LPFG* [28] introduces several changes to an L-system's expression which increase the efficiency of complex simulations.

3.3.1 User-defined parameter types

Instead of using simple types such as real numbers for parameters to the modules as in Examples 1 and 2, LPFG allows user-defined types to be passed as parameters. As such, a single parameter can store a long list of the module's properties. User-defined types allow large quantities of information to be made available as context, without the need for prohibitively complex production headers.

3.3.2 Fast information transfer

According to the original definition of context-sensitive L-systems, the context in a production comes from the state of the module at the previous derivation step. This information is, by nature, one step out of date. Thus, to propagate a signal across a branch of length n would require n rewriting steps. This results in very slow signal propagation, as depicted in Figure 3.3.

The new L-system framework LPFG [29] now allows for *fast information transfer*, which can propagate this same signal in a single rewriting step. Instead of using context from the previous rewriting step, the simulation can access already-interpreted neighbour modules for the current step to obtain up-to-date context information. Using this *new context* enables the propagation of a signal across the entire branch, regardless of its length, in a single rewriting step, as depicted in Figure 3.4. The use of new context in LPFG productions is indicated by the symbols '«' or '»'.

However, this new context can be obtained only in one direction, according to the order in which the modules in the string are rewritten. Although in concept, all the modules in an

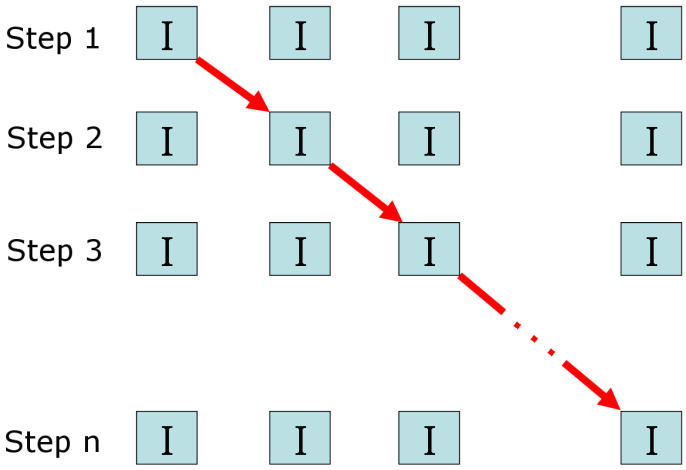


Figure 3.3: Signal propagation using old context

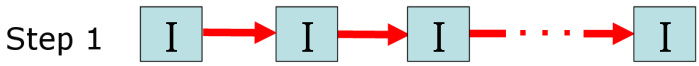


Figure 3.4: Signal propagation using new context

L-system are rewritten in parallel, computation in fact proceeds along the string by rewriting one module at a time. Using LPFG, the direction of rewriting can be set as left-to-right (*Forward*) or right-to-left (*Backward*), thereby setting the direction from which new context can be obtained. Thus signals can be propagated quickly in either direction across the string, speeding up many simulations. In fact, interactive rates are obtained for the growth simulations of all images presented in this thesis. Precise measures of simulation time are given in Chapter 6.

The use of L-systems to express biomechanical models of tree growth is advantageous as it allows rapid flow of information throughout a growing structure. However, because L-systems can only propagate information forward and backward along branched structures and not in 3 dimensions, the level of detail representable by the model is restricted. The smallest unit of consideration in this model is the segment or internode; a more fine-grained finite element representation considering three-dimensional components of a segment is not

possible using only L-systems. As such, some simplifications must be made in the biomechanical representation so that each segment of a tree stem can be treated as a unit. These simplifications are justified by the visually pleasing results and interactive running times of the simulations described in this thesis.

Chapter 4

Tree growth and resource allocation

A tree's growth governs its resulting architecture. To produce realistic tree architectures for use in biomechanical simulation, several aspects of tree growth are simulated.

The first aspect of the growth simulation to be described reproduces the yearly cycle of seasons. As will be explained in Section 4.1, the division of growth into seasons affects the results of the biomechanical simulation.

Secondly, a tree's means of control over its architectural development is simulated. Examination of real trees shows that some portions of trees are more likely to grow vigourously than others. Vigourous branches are more likely to grow longer and produce more child branches than others of low vigour. Vigourous buds are more likely to produce shoots rather than to abort.

Two approaches to controlling these aspects of architecture are presented in this chapter: a descriptive approach based functions of positional information, and a model of resource allocation symbolic of nutrient flow within a tree. These are presented in Sections 4.2 and 4.3 respectively.

4.1 Simulation of seasons

A tree's growth, and consequently its resulting architecture, is tightly linked to the cycle of seasons. In temperate climates, tree growth is a cyclical process of shoot elongation (spring), bud formation (summer), a bud-cooling period during which there is no primary growth (winter), and finally the shooting of some buds and aborting of others during the following spring [53].

To produce more realistic tree shapes, a tree's natural cycle of development is replicated

by incorporating this seasonal pattern into the simulation.

4.1.1 Seasons in the model

During the spring and summer seasons, both primary and secondary growth occur. Existing apical meristems produce a sequence of internodes and nodes which include a bud, as well as a leaf and possibly a fruit. Each newly created bud has the potential to become a shoot during the following year. The leaves and fruit have mass, which adds to the supported weight of the parent internode and thus affects the biomechanical simulation.

Secondary growth in the form of radial tree rings occurs as a result of the above-described primary growth, according to the pipe model of Equation 2.2:

$$r_{parent}^{P} = r_{lateral}^{P} + r_{straight}^{P} \tag{4.1}$$

where r is the cross-sectional radius and P is a positive constant, observed to be between 2.49 and 3 for most trees [34]. Equation 4.1 applies to the case of bifurcation in which a parent segment has produced a lateral child at some deviation angle, as well as a straight child which lies in the same direction as the parent.

Tree rings in nature can be produced only at a certain rate, so that not all of the radial growth demanded by the pipe model can occur at once. The pipe model calculation determines the new goal for radial growth r'_{goal}, and during each step of growth a certain percentage of that goal is achieved,

$$r'_{parent_{goal}} = \sqrt[P]{r_{straight}^{P} + r_{lateral}^{P}} \tag{4.2}$$

$$r'_{parent} = r_{parent} + \kappa(r'_{parent_{goal}} - r_{parent}) \tag{4.3}$$

where $\kappa \in [0,1]$ is the proportion of the required radial growth which may be accomplished during the current time step.

The fact that rings do not form instantaneously in support of new distal branches actually aids the process of branch deformation. Since radial growth is not immediate, branches

will sag under the new weight of added distal segments before radial growth reinforces their existing shape [15]. The slower the rate of radial growth, the more deformation occurs in the branches. This ring-production rate κ is a user-definable parameter in the model.

The fruit and leaves produced by the model also grow in size and mass throughout the summer. When a fruit or leaf is created, it is given a goal mass $m_{fruit_{goal}}$ or $m_{leaf_{goal}}$. This goal is a stochastic value centered about a user-specified average mass for fruit or leaves. Growth is accomplished by increasing the mass at each step according to the functions

$$m'_{fruit} = m_{fruit} + \kappa(m_{fruit_{goal}} - m_{fruit}) \quad (4.4)$$

$$m'_{leaf} = m_{leaf} + \kappa(m_{leaf_{goal}} - m_{leaf}). \quad (4.5)$$

The size of the fruit or leaf increases correspondingly, according to a user-specified density value for each. κ is defined as it was in Equation 4.3.

Summer ends when all primary growth has been completed. Note that if $\kappa < 1$, branches will still be undergoing radial growth after summer ends.

At the end of summer, leaves and fruit are shed and the tree branches spring back up slightly because of the reduced load. This effect is depicted in Figure 4.1.

Figure 4.1: A fruit tree shown in summer, bearing the load of fruit and leaves, and the following winter under reduced load.

During winter, the production of radial rings is completed. This increase in girth adds to

a branch's rigidity, reinforcing its existing shape. Another factor contributing to the rigidification of branches during winter is the aging of the wood, as will be described in Chapter 8.

After winter has ended, the fates of the previous year's buds are decided. A bud either shoots or aborts, depending on the results of the architecture control simulation. This simulation also determines how many internodes the new shoot will have. In the following sections, two methods of controlling architectural development are described.

4.2 Descriptive rules for architecture development

The first method of controlling architecture is descriptive; it specifies rules for tree growth based on observations of natural trees. For example, Wilson has noted that higher-order branches elongate more slowly than branches of lower order [53]. This phenomenon is called *acrotony*, and can be built into the model in the form of a stochastic rule governing growth.

A number of such rules are created using L-studio's graphically defined functions [28]. The functions allow for user control over features of tree architecture, and are based on positional information in the model. The interface for the function editor in L-studio is depicted in Figure 4.2.

The output of each function is used to set the value of an architectural element in the model. In this manner, two or more architectural traits can be made to influence each other. For instance, one possible representation of acrotony links internode length to branching order. The model represents this rule as a function f,

$$\text{internode length} = \rho * f(\text{ order }) \tag{4.6}$$

which the user can control graphically by moving control points. The value ρ is a random number which ensures a stochastic outcome, so that many different individual trees can be

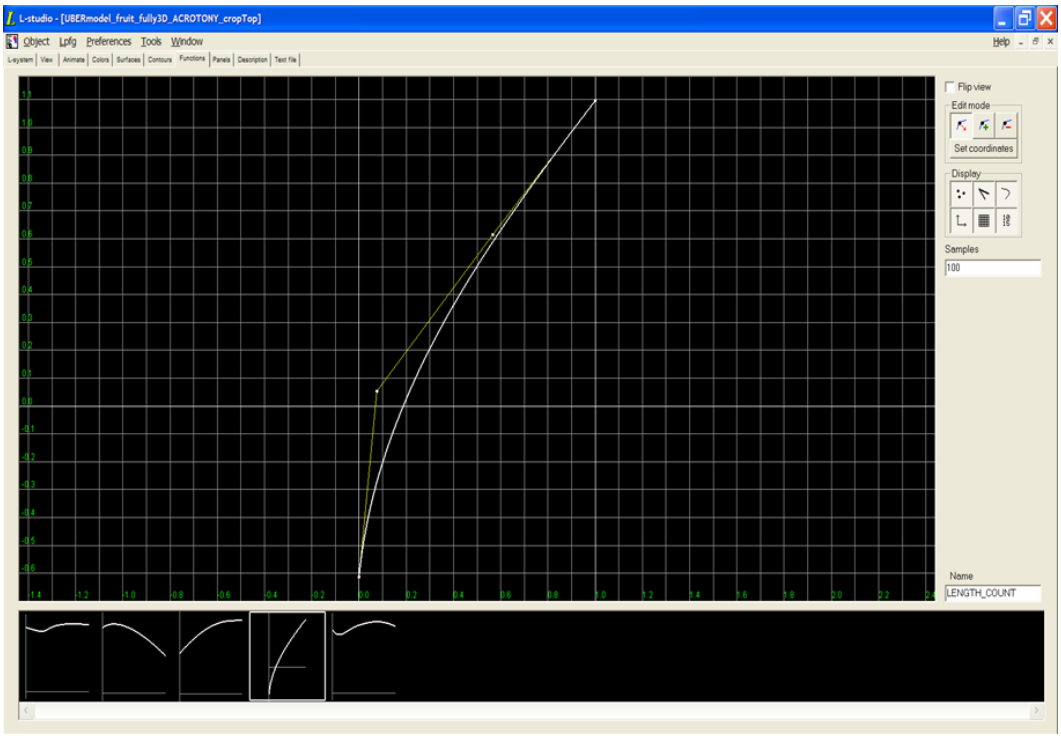


Figure 4.2: L-studio's graphical function editor provides control points to be modified by the user

created from the same tree species.

The acrotony that develops in the model tree from the use of this function is depicted in Figure 4.3. The first image shows the buds in a year's growth, with bud numbers increasing distally. The second image shows the tree in the next year, after the buds have either formed shoots or aborted. As seen in this image, the higher-numbered buds produce the longest shoots. The third image shows the tree after several more years of development.

Many other functions such as the Equation 4.6 can be used to control different aspects of tree architecture. Often, more than one parameter controls an outcome. For instance, the fate of a bud depends on several factors:

$$\text{bud shoots} = \rho * (f_1 (\text{height in year's growth}) * f_2(\text{order}) * f_3(\text{parent length})) > 1) \tag{4.7}$$

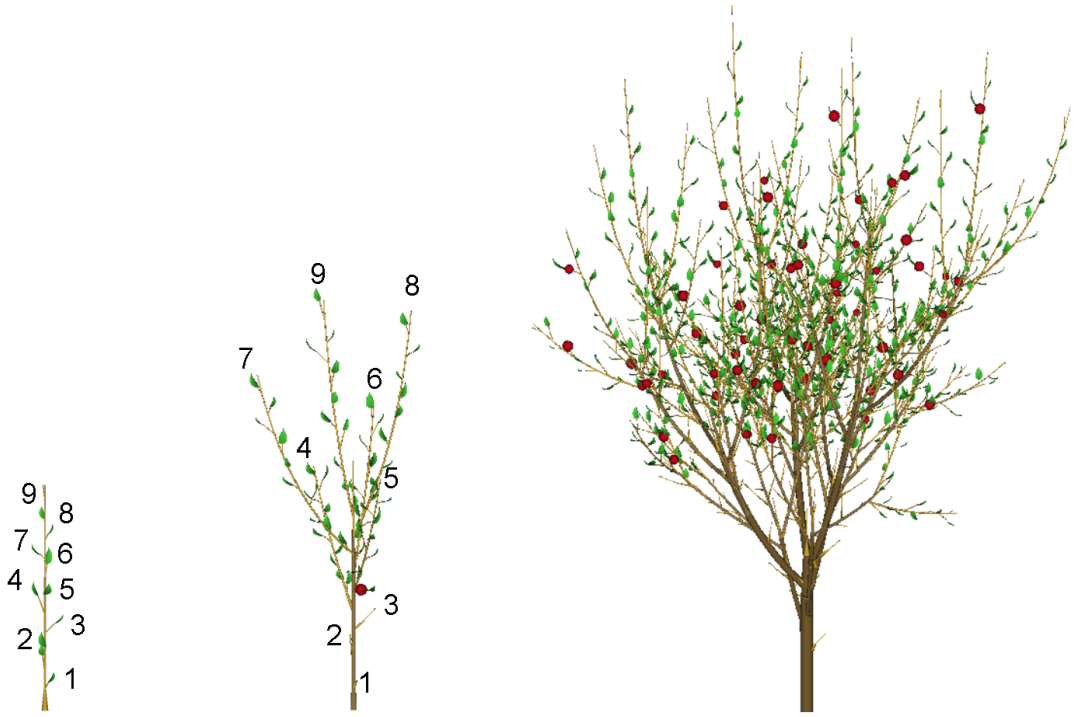


Figure 4.3: Acrotony is achieved from a graphically-defined function

Manipulating the control points of a model's graphical functions allows users to represent different tree architectures without altering the L-system code. Figure 4.4 shows two trees of different architectures created by modifying these functions.

4.3 Allocation of resources

Instead of creating descriptive rules to reproduce botanical observations, the second method of architectural control emerges from a physiologically-based simulation of resource allocation. This method is symbolically based on the flow of water and nutrients throughout a growing tree, though it is much simplified.

In botanical trees, the allocation of resources such as carbohydrates and water plays a role in determining which portions of the tree will become vigourous and which will be shed [53]. In a developmental model, growth can be controlled by simulating resource propagation throughout a tree. Hart et al. [21] have simulated the allocation of *growth resources*

Figure 4.4: Two trees of different architectures created by modifying graphical functions

throughout the tree, which control how much each limb will lengthen. Resources were allocated according to the ratio of the computed *growth rate* of the particular branch as compared to that of its subtree.

A more rigourous physiologically accurate simulation of resource allocation was presented in [2]. Allen et al. presented the L-PEACH model, a detailed physiological simulation of carbon flow throughout a growing peach tree. This work simulated the effects of sources and sinks, and distribution followed an analogy to electric circuits. Organs in the model were shed if they did not receive enough nutrients.

The resource allocation model of Borchert and Honda (BH) [8] is a more simple simulation of water and mineral flow in a growing tree. In this model, resources flow *acropetally* from the base of the tree to the apices. At bifurcation points, the incoming resources are divided in proportion to the number of apices supported in the subtree of each competing branch. If the quantity of resources that arrive at a certain apex is larger than a threshold value, another bifurcation occurs. Prusinkiewicz et al. extended the model of Borchert and Honda to simulate root-shoot interactions in a growing plant, within the context of L-systems [38].

The resources allocation model presented in this thesis is based on the model of Borchert and Honda, as it is a simple strategy which will not greatly reduce the speed of the bio-mechanical simulation. In my model, resources are distributed throughout the tree and are used to determine which branches become vigourous. At non-branching points, a segment receives the entire quantity of resources that has passed through the segment below it. At branching points, the incoming resources are divided among child branches according to a distribution heuristic described below.

A parent's resources should be divided according to the relative sizes of each child's supported subtree. Borchert and Honda determined these sizes by counting the apices present in each subtree. However, my simulation's use of the pipe model to determine the girths of branches allows a simpler approach. Recall from Chapter 2 that the girth of a branch is determined as

$$r_{parent} = \sqrt[P]{r_{straight}^P + r_{lateral}^P}. \tag{4.8}$$

Because Equation 4.8 essentially calculates the girth of a branch based on the size of the subtree it supports, the relative girths of child branches may be used to determine the division of resources at branching points, as depicted in Figure 4.5.

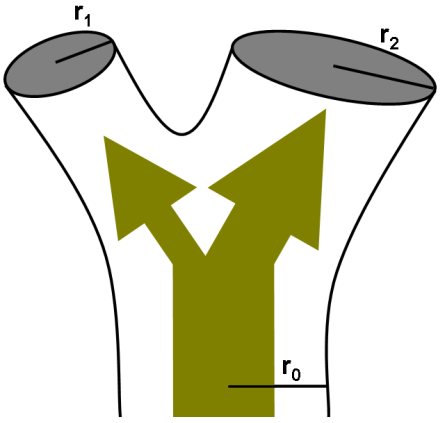


Figure 4.5: At branching points, resources are distributed according to relative girth.

A manipulation of Equation 4.8 yields

$$\frac{r^P_{straight}}{r^P_{parent}} + \frac{r^P_{lateral}}{r^P_{parent}} = 1. \tag{4.9}$$

Because their values sum to 1, the two terms $\frac{r^P_{straight}}{r^P_{parent}}$ and $\frac{r^P_{lateral}}{r^P_{parent}}$ can be used as proportions to divide the parent's resources between straight and lateral children:

$$V_{straight} = \frac{r^P_{straight}}{r^P_{parent}} * V_{parent} \tag{4.10}$$

$$V_{lateral} = \frac{r^P_{lateral}}{r^P_{parent}} * V_{parent}. \tag{4.11}$$

This strategy conserves the amount of resources flowing through the tree because the equations perfectly divide incoming resources at every branching point. It also distributes resources evenly among buds along a shoot, as will be explained in the next section.

The total amount of incoming resources in the model is also determined from the girth of the base;

$$V_{base} = (K\, r_{base})^d, \tag{4.12}$$

where r_{base} is the cross-sectional radius of the base of the trunk, K and d are positive constants which can be adjusted to obtain a more or less vigourous tree. K controls the vigour of the tree at the beginning of its growth, and d controls growth in vigour as the tree develops.

4.3.1 Fate of buds

Whereas the BH model decides the fate of each bud at every simulation step, my model simulates the natural cycle of seasons, producing a series of buds in summer and waiting through winter until the following spring to determine the fate of these buds. Thus, when spring arrives there are a number of buds along each shoot which must compete with each other for available resources.

Acceptable distribution of resources among buds is very important to the shape of a developed tree, because a bud's vigour determines whether it will shoot or abort during

the next year. Though not all buds of real trees are equally likely to shoot, an equitable distribution of resources to buds provides a good base for the model, from which deviations may be made when modelling specific tree species.

Because the pipe model controls both cross-sectional radii and resource allocation, using Equations 4.10 and 4.11 to distribute resources along the shoot will ensure that incoming resources are distributed evenly among buds. All buds are defined to have the same girth, but the girth of the main branch decreases acropetally along a shoot because of the pipe model. Therefore, each successive distal bud along a shoot will obtain a larger *proportion* of its parent's resources, because the ratio $\frac{r^P_{lateral}}{r^P_{parent}}$ increases distally along the shoot. At the same time, the quantity of resources decreases along a shoot as portions are claimed by more proximal buds. The result is that the amount of resources in any segment is always directly proportional to its girth. As such the buds, being of equal girth, receive equal quantities of resources.

In nature, not all buds are equally likely to shoot. Depending on the tree being modelled, certain buds should be made more likely to shoot than others. To model effects such as acrotony, the allocation of resources must be modulated based on the height of the bud in the year's growth. This can be done with a graphically-defined function,

$$V_{lateral} = \frac{r^P_{lateral}}{r^P_{parent}} * f(h_{lateral}) * V_{parent} \tag{4.13}$$

$$V_{straight} = V_{parent} - V_{lateral}, \tag{4.14}$$

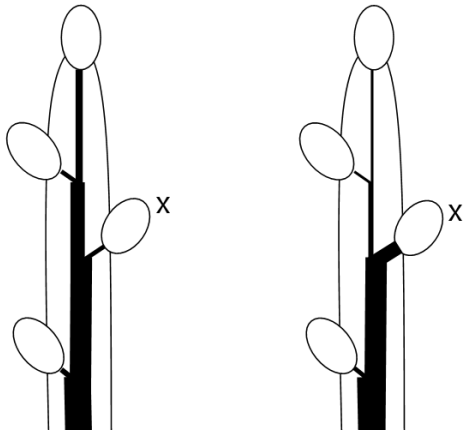
where $f(h_{lateral})$ is a function of the position of the lateral bud in the year's growth when the buds are counted acropetally. Use of this function in Equation 4.13 alters the pipe model distribution, so Equation 4.14 must explicitly calculate the remaining resources to ensure that the quantity of resources is preserved.

To determine which buds shoot, computation proceeds along each shoot in the acropetal direction. At each bud in turn, the quantity of resources to be allocated is computed according to Equations 4.10 and 4.11 (or 4.13 and 4.14). If the result exceeds a threshold, the bud is

marked for shooting during the approaching summer. The number of internodes that this new shoot will produce is also determined at this time, and is proportional to the bud's assigned resources V:

$$n = (f(V)) \tag{4.15}$$

where n is the number of internodes to be produced by the new shoot and f is a graphically defined function modifiable by the user.



(a) (b)

Figure 4.6: Resources flowing to several buds. In (a), none of the buds are determined to shoot. In (b), bud X is determined to shoot and becomes a sink for resources.

After a bud is marked for shooting, it becomes a *sink* for resources, as depicted in Figure 4.6. This sink removes an even greater quantity of resources from the stream than the bud has been assigned, in preparation for the production of a new shoot. The quantity of resources claimed by the sink is proportional to the number of internodes n that will be in the shoot, as determined from Equation 4.15,

$$V_{parent} = V_{parent} - (\, V_{lateral} \, * \, \varsigma \, n), \tag{4.16}$$

where $\varsigma \in (0,1)$.

Because this sink has claimed such a great quantity of resources, buds further along the current shoot will have fewer resources to be distributed among them and will be less likely to

shoot. In this way, competition between buds is simulated, although buds in more proximal positions along the branch are clearly favoured. A more complex method which compares the vigour of all buds simultaneously would result in a fairer competition between buds, but for the purposes of this model the above simulation is adequate. This method allows the fate of all buds to be determined in a single forward pass of an L-system simulation.

4.3.2 Shedding of unproductive branches

A branch is productive as long as it is undergoing primary growth somewhere in its supported subtree. As in a real tree, branches in the model that are no longer productive are eventually shed.

Productivity can be gauged by measuring secondary growth. If an internode is no longer undergoing secondary growth, according to the pipe model it cannot be supporting any new primary growth. A branch is shed it if does not produce secondary growth within a specified number of years after having been marked as unproductive. The number of years before shedding is a user-definable parameter of the model.

When an unproductive branch is shed, the resources it had previously claimed become available for redistribution throughout the rest of the tree. Thus, a significant culling of unproductive branches results in a surge in primary growth during the following year.

When branches are shed, strict adherence to Equation 4.2 would reduce the cross-sectional radius of supporting branches. Such thinning of branch girth is not observed in nature and must be avoided by the model. The equation for cross-sectional radius goal of an internode (to replace Equation 4.2) is therefore

$$r'_{parent_{goal}} = Max\left(\sqrt[P]{r^P_{straight} + r^P_{lateral}} \;,\; r_{parent_{goal}} \right) \tag{4.17}$$

indicating that this goal cannot decrease from one iteration to the next.

4.3.3 Fairness of resource allocation

To demonstrate the effect of resource allocation on controlling tree architecture, the visual results of my original resource-distribution algorithm will be compared with those of a modified strategy.

A *fair* distribution strategy will apportion resources at bifurcation points in proportion to relative girth, as described in Equations 4.10 and 4.11. This policy can be altered so that branches already favoured receive an even larger share of the resources in a winner-takes-all manner. To shift toward such a strategy, the allocated resources are skewed using variable $F \in [0,1]$:

$$\sigma_{small} = F * \sigma_{small} \tag{4.18}$$

$$\sigma_{large} = 1 - (F * \sigma_{small}) \tag{4.19}$$

where σ_{small} and σ_{large} are the resources assigned by the fair strategy to child branches of smaller and larger girth, respectively.

The model provides a slider which allows a user to modulate this value F. Figure 4.7 depicts the difference between the fair ($F = 1.0$) and a winner-takes-all strategy ($F = 0.2$) on otherwise identical models. The fair strategy results in a full tree with many small branches, whereas the winner-takes-all strategy creates a tall tree with long, sparse branches.

Production rules for the model of resource allocation are given in the Appendix.

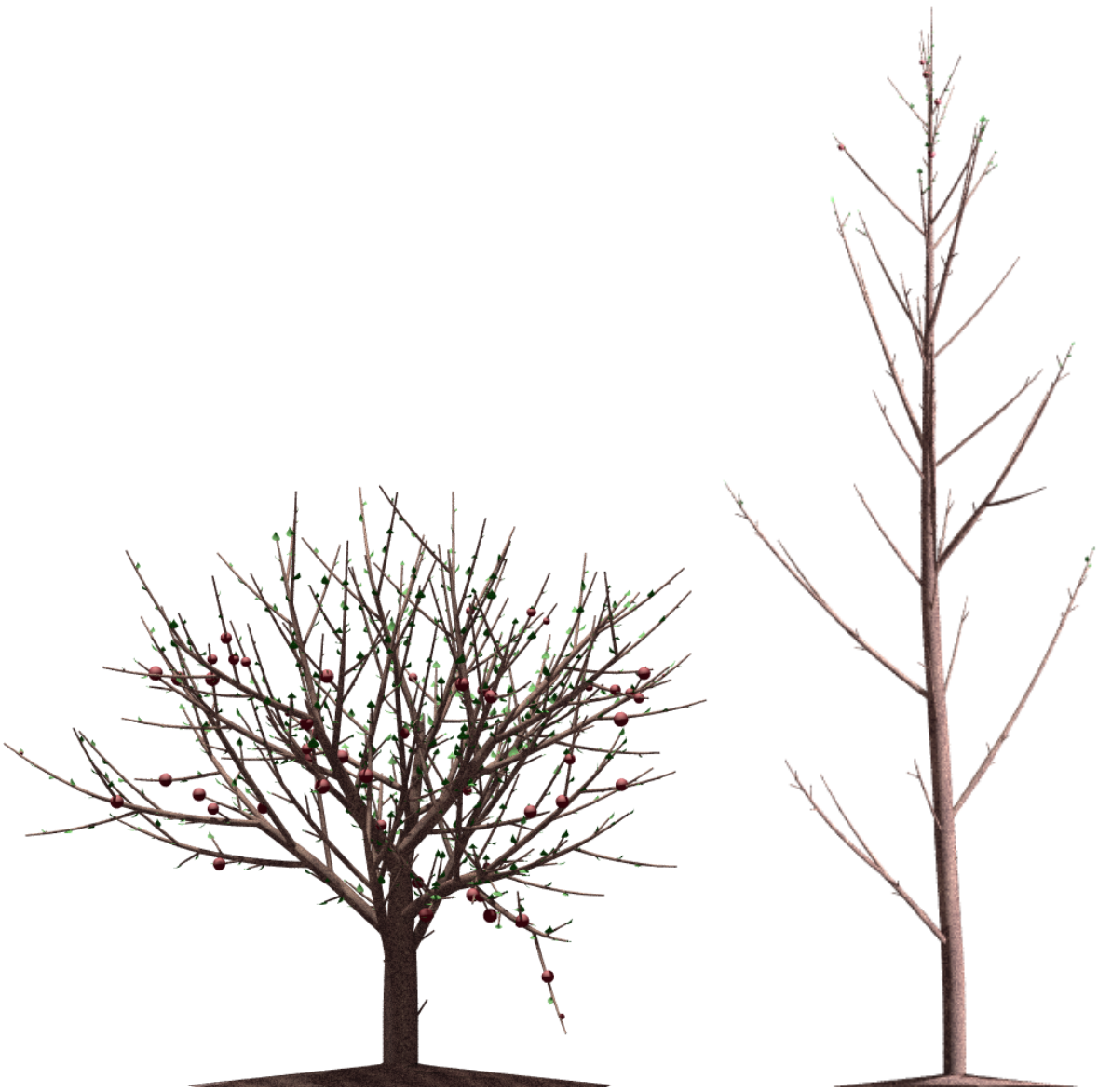


Figure 4.7: A tree model using two different resource allocation strategies, (a) $F = 1$ and (b) $F = 0.2$.

Chapter 5

Elasticity Theory

This section provides an introduction to the theory of elasticity as it relates to biomechanical tree models. In particular, we are interested in the application of elasticity theory to a simulation of bending.

5.1 Load and stress

An object constructed from an elastic material bends or otherwise deforms temporarily in response to a *load*, or applied force. Load is is measured in *Newtons*, N, where $1N = 1kg\ m\ s^{-2}$. An *axial load* acts along the long axis of an object, and a *lateral load* acts parallel to the cross-section. Any load can be divided into its axial and lateral components [35].

In elasticity theory, a structure can be classified according to the type of load it bears. A *rod* bears axial load and responds by positive or negative elongation (tension or compression) [1]. When a rod is bent, infinitesimal portions of it undergo elongation. Portions of the rod above a bend experience tension, and portions underneath experience compression (Figure 5.1). Though infinitesimal portions elongate, the rod as a whole maintains a fixed length.

A rod experiences a *stress* due to this load, of a magnitude inversely proportional to the area being loaded:

$$\sigma = \frac{F}{A}, \tag{5.1}$$

where F represents the magnitude and direction of the load, and A is the area of the plane being loaded. Stress is measured in *Pascals*, or $N\ m^{-2}$. In the general case, σ is a tensor, as its value depends on the orientation of the plane undergoing loading and the direction of the force causing load [9]. However, in the case of a rod, the orientation of the plane being loaded is considered to be parallel to the cross-section. In cases where the orientation of the

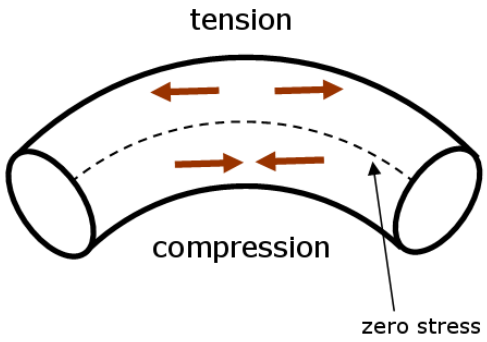


Figure 5.1: A bending rod undergoing tensile stress above and compression stress below the neutral axis, where no stress is experienced.

plane being loaded is known, Chou et al. note that σ is reduced to a vector whose direction is the same as that of the force causing load [9].

5.2 Young's modulus and strain

Two objects of identical shape undergoing the same stress may react differently because of the properties of the material from which they are made. Some materials bend more easily than others, and this is reflected in their material stiffness constant E, called *Young's modulus of elasticity*. If a material is *isotropic*, E is constant in all directions of measurement. For *anisotropic* materials, however, material properties such as Young's modulus relate to a specific direction of measurement. Wood is anisotropic, having different material properties when measured along the grain, or perpendicular to the grain in directions radial or tangential to the tree rings [35]. For purposes of simplicity, however, wood is assumed to be isotropic in my model. This assumption is also made in [33].

Using the stress measurement σ and the value of E for a material, one may predict how far an object will deform. For clarity, consider the one-dimensional case of a slender rod experiencing normal stress σ parallel to its long axis. The *strain* ε is then the deformation

per unit length:

$$\varepsilon = \frac{\Delta l}{l_0}, \tag{5.2}$$

where l_0 is the unstrained length of the rod.

There exists a linear relation between stress and strain, called Hooke's Law:

$$\varepsilon = \frac{\sigma}{E}. \tag{5.3}$$

This linear relationship is what defines elastic, or *Hookean*, materials. In objects made from elastic materials, stress from a load causes a linearly proportional deformation as in Figure 5.2. The bending depicted by my biomechanical model occurs within the elastic deformation range; as soon as the load is removed, the object returns to its initial state.

When the applied force exceeds a certain limit, however, the material loses its elastic properties and deforms permanently. Beyond this point, the relationship between stress and strain is no longer linear: reducing stress may not reduce the deformation (Figure 5.2). The point where the nonlinearity begins is called the *yield point*, and marks the beginning of the material's *plastic* phase. The yield point is specific to the particular material and the type of load acting on it.

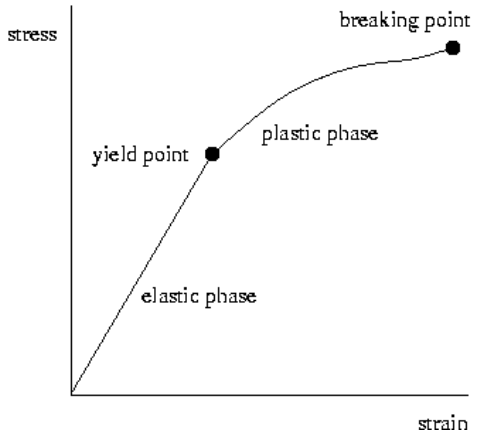


Figure 5.2: Stress vs. strain graph showing the elastic and plastic phases of a material

With an even further increase in stress, the material will reach a *breaking point* at which it fractures or fails (Figure 5.2). The stress σ at this point is called the *ultimate stress* for the

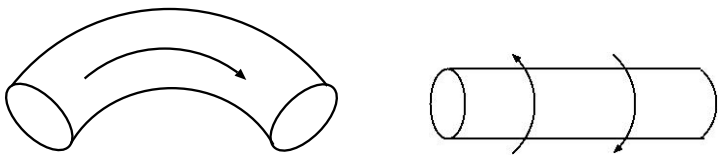
material and type of load.

5.3 Strength

The *strength* of a material is the maximum stress that the material can withstand before breaking (*breaking strength*) or becoming plastic (*yield strength*) [35]. A material's strength differs under each type of stress. Wood is weakest under compression; its compression strength is half of its strength under tension [33]. Consequently, wood fails first under compression on underside of a downward-bending branch [33]. Niklas provides an estimate of 0.1 GPa for tension strength of wood along its grain [35]. Therefore the compression strength will be half this value, or 0.05 GPa.

5.4 Rigidity and moment of area

The *rigidity* $\mathfrak{R}$ of an object is its resistance to deformation, which depends on both the material's stiffness and the object's shape. An object shows different resistance to different types of deformations. In particular, resistance to bending, or *flexural rigidity*, differs greatly from resistance to twist, or *torsional rigidity*.



(a) (b)

Figure 5.3: Flexural and torsional rigidities

5.4.1 Flexural rigidity – resistance to bending

Flexural rigidity $\Re_f$ is resistance to bending along the rod's long axis, as in Figure 5.3(a). The flexural stiffness is specified by its Young's modulus E. For the purposes of calculation, an object's shape is described by its *second moment of area* I. This is a measure of the distribution of material within an object with respect to its centroid axis, and is based on the cross-sectional shape of the object and the direction of measurement [33]. In general, an object's resistance to bending depends on the direction in which force is applied. However, for a rod with circular cross-section (so that I is constant in any direction) composed of an isotropic, homogeneous material (so that E is constant), flexural rigidity reduces to a scalar. Though the assumptions of constant E and I for a wood stem reduce the model's fidelity to nature, they simplify many calculations.

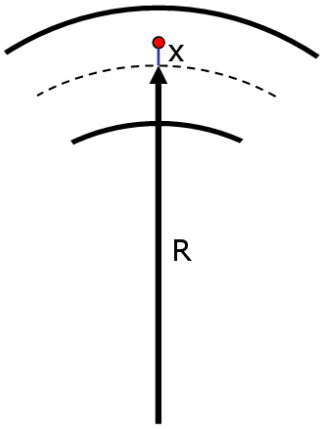


Figure 5.4: Distance x from the neutral axis, and radius of curvature R.

Following the derivation in [35], the equation for flexural rigidity $\Re_f$ is formulated, beginning with the observation that strain ε experienced at any location along a bending rod can also be defined as

$$\varepsilon = \frac{x}{R}, \tag{5.4}$$

where R is the radius of curvature and x is the distance from the *neutral axis*, the axis at which there is no stress or strain as a result of bending (see Figure 5.4). Using Equation 5.3,

it follows that

$$\sigma = \frac{Ex}{R}, \tag{5.5}$$

or

$$\sigma = \kappa E x, \tag{5.6}$$

if $\kappa = \frac{1}{R}$. κ is called the *curvature of bending* [35]. Making use of Equation 5.1 yields

$$F = \sigma A = \kappa E x A. \tag{5.7}$$

The magnitude of torque, M, caused by force F acting across distance x, therefore equals

$$M = Fx = \kappa E x^2 A. \tag{5.8}$$

Equations 5.4 to 5.8 consider only one location along a bending rod. The total bending moment $\overline{M}$ is found by integrating over the entire area of the cross-section, A:

$$\overline{M} = \kappa E \int_A x^2 dA = EI\kappa \tag{5.9}$$

The first two terms in Equation 5.9 define the flexural rigidity of the rod,

$$\Re = EI. \tag{5.10}$$

Thus, Equation 5.9 can be rewritten as

$$\overline{M} = \Re\kappa, \tag{5.11}$$

which indicates that the same moment $\overline{M}$ causes different amounts of curvature κ when acting upon objects with different flexural rigidities $\Re$.

Values of I for objects with common cross-sections are given in reference texts such as [35]. For a circular cross-section,

$$I_{circle} = \frac{\pi}{4} r^4, \tag{5.12}$$

where r is the cross-sectional radius. This equation holds for any axis of bending parallel to the cross-section.

5.4.2 Torsional rigidity – resistance to twist

Torsional rigidity $\mathfrak{R}_t$ is an object's resistance to twist around its long axis, as in Figure 5.3(b). It is defined using an equivalent derivation to that of flexural rigidity. The resulting equation is similar, but uses the equivalent constants for the case of torsion:

$$\mathfrak{R}_t = GJ, \tag{5.13}$$

where G is the *shear modulus* of the material and J is the *torsional constant* [35]. For circular cross-sections, the torsional constant is the same as the *polar second moment of area*, which is simply the sum of the moments of area in two orthogonal directions along the cross-section [35]. Thus, for a circular cross-section,

$$J_{circle} = I_{circle} + I_{circle} = \frac{\pi}{2} r^4. \tag{5.14}$$

5.5 Bending stress

The biomechanical model presented in this thesis deals particularly with deformations due to bending. Bending moment, or torque, causes bending stress, which appears as tensile stress at the top of the bend and compressive stress on the underside, as illustrated in Figure 5.1.

For a rod, bending stress is quantified as

$$\sigma_B = \frac{Mx}{I}, \tag{5.15}$$

where M is the bending moment, I is determined by Equation 5.12 and x is the distance from the neutral axis, in the plane of bending [33]. Thus bending stress increases with distance from this axis, and is maximized at the edge of the cross-section. A branch being bent will experience high compression stress on one side, which decreases to zero stress at the centroid axis. On the other side of the axis, increasing tensile stress is maximized on the under side of the bend. Thus, for a circular cross-section, Equation 5.15 is maximized when the distance from the centroid axis is maximized – that is, when $x = r$, the cross-sectional radius.

From Equation 5.15, and given a value of breaking strength, the bending moment M that causes breakage can be calculated[33]. Breakage is discussed further in Section 8.6.

The stiffness of a material can be measured specifically for bending, E_B. For example, the stiffness of poplar wood under bending is $E_B = 10.9$ GPa [16]. This measure includes the effects of shearing [16].

5.6 Shear stress from torsion

The shear stress from torsion can be calculated by an equation equivalent to 5.15:

$$\tau = \frac{M_T x}{J} \tag{5.16}$$

$$= \frac{2M_T x}{\pi r^4} \text{ for a circular cross-section.} \tag{5.17}$$

In this equation, M_T is the torsional moment, or torque around the stem axis. Equation 5.17 is maximized when $x = r$, the cross-sectional radius.

Chapter 6

Biomechanics

A tree's final shape is greatly affected by interaction with its environment during growth, because a change in shape at any stage affects the course of its future development. In this chapter, a biomechanical simulation of tree growth is described.

The simulation process grows the tree by periodically adding longitudinal segments to the tips of the branches, with their initial orientations taking into account the effects of negative geotropism. The mass of each new segment increases the torque acting on previous branches. More strength is required to handle this additional weight, so the existing branches must build themselves up radially in order to support the extra load. This adds to their girth and thus their stiffness, which affects the curvature of the branches. After the addition of each new segment, the simulation performs a series of relaxation steps so as to converge to static equilibrium for the current structure. Thus at any stage of growth, the equilibrium shape of the tree may be observed.

In Section 6.1, Jirasek's work on biomechanical modelling of unbranched plant axes or *stems* is reviewed. In Section 6.2, an extension to her method is presented to correctly simulate biomechanics at branching points, so that entire trees may be represented [51].

6.1 Biomechanical model of a single stem

Jirasek et al. conceptualized an individual plant axis as an elastic rod which can bend, but not stretch [25]. She modeled the bending of a stem by rotating a set of joints between small straight segments or *links*. This representation was inspired by the concept of a *mechanical manipulator* robot described by Craig [10], depicted in Figure 6.1. The model grows by appending new segments to its free end.

By convention, the links are indexed as l_i, with i from 0 to n; l_0 is the root or *proximal* link, and l_n is the *distal* link, which is the tip of the stem (Figure 6.1). Link i is called the *parent* of link $i+1$. Each link's position and orientation are specified in relation to its parent. For simplicity, and because the segments are very small, the mass m_i of each segment is assumed to be located at its distal end. Between each pair of segments is a rotational joint with three degrees of freedom. These joints account for the bending of the stem.

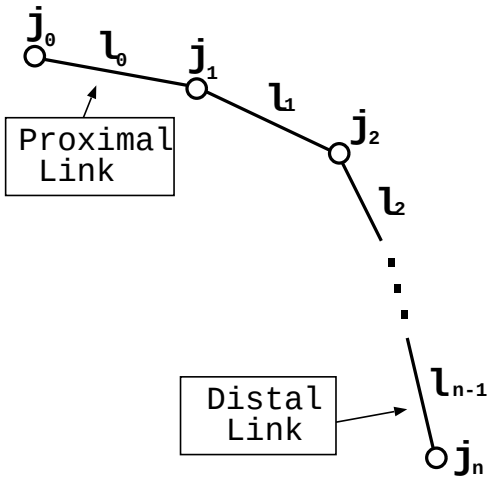


Figure 6.1: A mechanical manipulator with straight links and rotational joints.

To describe the relative properties of segments, a three-dimensional orthogonal reference frame is assigned to each segment. This reference frame consists of the axes *Head* ($\vec{H}$), *Left* ($\vec{L}$) and *Up* ($\vec{U}$) described in [41]. The long axis $\vec{l}$ of each segment always lies along its $\vec{H}$ vector. Thus, the $\vec{H}\vec{L}\vec{U}$ frame rotates along with the link.

Although the stem model is made up of discrete segments, each segment is small enough that the entire stem can be conceptualized as a continuously curved rod [39]. Thus, instead of defining angles of rotation between segments, the model uses a rate of rotation per unit length, $\vec{\Omega} = \frac{d\vec{\theta}}{dl}$.

Because segments are assumed to be very short, joint rotations between segments are therefore infinitesimal. These infinitesimal rotations can be expressed as vectors [1]. Consequently, the *rates* of these infinitesimal rotations can also be stored as vectors, as they will be

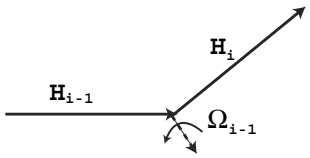


Figure 6.2: New heading vector direction found by rotation around $\vec{\Omega}$

used only in combination with small segment lengths. Thus $\vec{\Omega}$ can be expressed as a vector, whose direction gives the axis of rotation for a joint, and whose magnitude gives the angle of rotation, in radians per unit length of the segment.

Along a single axis, the child segment's orientation $\vec{H}_i$ is determined from the parent's orientation $\vec{H}_{i-1}$ using the rotational velocity vector $\vec{\Omega}_{i-1}$ of the parent link. Jirasek et al. performed this rotation using the vector cross-product, as described in [41]:

$$\vec{H}_i' = \vec{H}_{i-1} + \vec{\Omega}_{i-1} \times \vec{l}_{i-1}, \tag{6.1}$$

where $\vec{l}_{i-1}$ is the long axis of the segment. Using the cross-product was possible because the authors represent the rotational velocity $\vec{\Omega}_{i-1}$ as a vector. My model will perform all rotations using quaternions, so as to handle the non-infinitesimal rotations at branching points. The details of this approach are described in Section 6.2.

Jirasek's approach also defines a rest, or preferred, rotational velocity $\underline{\vec{\Omega}}$ [39]. $\underline{\vec{\Omega}}$ is the stem's rate of rotation in the *unloaded state*, when it is not experiencing any torque due to gravity. The vector $\underline{\vec{\Omega}}$ is also used to account for the effects of *negative geotropism* in Jirasek's model. As defined in Section 2.4, negative geotropism is the tendency of a shoot to grow in the direction opposed to gravity. Clearly, the effects of self-weight and negative geotropism oppose each other: negative geotropism causes new growth to curve in an upward direction, whereas gravity pulls the existing stem downward. These competing influences, along with the branch's radial growth, create the sigmoidal shape predicted by Fournier et al. [15]. A stem under such conditions, produced by my model, is shown in Figure 6.7.

6.1.1 Torques due to negative geotropism and self-weight

The two effects of self-weight and negative geotropism are simulated by calculating the torques they exert on each segment. *Torque*, or the moment of a force, is the tendency of the force to cause a body to rotate about an axis [22]. It may be seen as the measure of rotational force applied to a joint, and is caused by the application of a force across a distance:

$$\vec{\tau} = \vec{l} \times \vec{F}, \tag{6.2}$$

where vector $\vec{l}$ represents the direction and distance across which $\vec{F}$ acts.

Jirasek calculated the gravitational torque $\vec{\tau}_i$ acting on each segment i due to the total overhanging mass M_i it supports. This overhanging mass is equivalent to the accumulated mass of all child segments, $M_i = \sum_{j=i}^{n} m_j$. It will be shown in Section 6.1.4 that $\vec{\tau}_i$ is equivalent to the torque due to the accumulated mass at the child segment, M_{i+1}, plus the child's accumulated torque $\vec{\tau}_{i+1}$ [25]:

$$\vec{\tau}_i = (\vec{l_{i+1}}) \times (M_{i+1}\vec{g}) \; + \; \vec{\tau}_{i+1}. \tag{6.3}$$

This torque causes a bend in the elastic rod by influencing the rotational velocity, $\vec{\Omega}$. Its influence depends on the rigidity of the rod:

$$\vec{\Omega}^{new} = \frac{\sum \vec{\tau}}{\Re}, \tag{6.4}$$

where $\Re$ is the rigidity of the segment, as defined in Section 5.4. When dividing by rigidity $\Re$, as in Equation 6.4, a distinction must be made between flexural and torsional rigidities. The $\vec{H}$ component is divided by the torsional rigidity $\Re_t$, to control the amount of twist about the $\vec{H}$ axis, and the $\vec{L}$ and $\vec{U}$ components are divided by the flexural rigidity $\Re_f$. Thus Equation 6.4 becomes:

$$\Omega_H^{new} = \frac{\tau_H}{\Re_t} \tag{6.5}$$

$$\Omega_L^{new} = \frac{\tau_L}{\Re_f} \tag{6.6}$$

$$\Omega_U^{new} = \frac{\tau_U}{\Re_f}. \tag{6.7}$$

The torque acting on joint *i* as a result of negative geotropism is determined using a force vector, $\vec{F^{tropic}}$ which indicates the direction (up) and the intensity of this tropism:

$$\vec{\tau}_i^{tropic} = \vec{l}_{i-1} \times \vec{F}^{tropic}. \tag{6.8}$$

This torque is used to initialize the rest rotational velocity $\vec{\underline{\Omega}}$ of a new segment:

$$\vec{\underline{\Omega}} = \frac{\vec{\tau}_i^{tropic}}{\Re} \tag{6.9}$$

Distinguishing between flexion and torsion, Equation 6.9 becomes:

$$\underline{\Omega_H} = \frac{\tau_{i_H}^{tropic}}{\Re_t} \tag{6.10}$$

$$\underline{\Omega_L} = \frac{\tau_{i_L}^{tropic}}{\Re_f} \tag{6.11}$$

$$\underline{\Omega_U} = \frac{\tau_{i_U}^{tropic}}{\Re_f} \tag{6.12}$$

6.1.2 Overview of the simulation

In summary, the biomechanical properties stored in each segment are:

- orientation relative to proximal neighbour, $\vec{H}\vec{L}\vec{U}$
- link vector $\vec{l}$
- mass *m* and accumulated distal mass *M*
- cross-sectional radius *r*
- flexural and torsional rigidities $\Re_f$ and $\Re_t$
- accumulated torque due to distal segments $\vec{\tau}$
- preferred rotational velocity $\vec{\underline{\Omega}}$
- acting rotational velocity $\vec{\Omega}$

Some of these segment properties depend on values from neighbouring distal segments, and some on values from proximal segments. For this reason, the model requires both *forward propagation* (accumulating information toward the distal end of the stem) and *backward propagation* (toward the proximal end), to obtain information from both directions. Sections 6.1.3 and 6.1.4 examine each direction of propagation in detail. The model performs forward and backward passes in alternation until equilibrium is reached. In this manner, values calculated in one direction influence those in the other direction. When static equilibrium is reached, growth is simulated by the addition of new segments to the model, as will be described in Section 6.1.5.

6.1.3 Forward propagation

According to the model of the mechanical manipulator described in Section 6.1, the child segment's orientation is inherited from the parent with some adjustment due to forces in the system. Specifically, the joint at which the child frame is located is rotated by the rotational velocity $\vec{\Omega}$ of the parent link. This rotation is performed according to Equation 6.1 in Jirasek's system. In Section 6.2 the use of quaternions to perform this rotation will be described.

6.1.4 Backward propagation

Most of the physical information must be accumulated backward along the stem, from child segment to parent segment. A segment i has a gravitational torque $\vec{\tau}_i$ acting on it due to the accumulated masses M_i of all its child segments. Recall the torque equation

$$\vec{\tau}_{i-1} = \vec{l}_i \times \vec{F}_i, \tag{6.13}$$

where $\vec{l}_i$ is the distance across which $\vec{F}_i$ acts. Also recall that masses are located at each joint, represented by the circles in Figure 6.3. The simulation requires accumulation of torques

due to gravity acting on all distal mass points. Figure 6.3 depicts the torques that need to be summed.

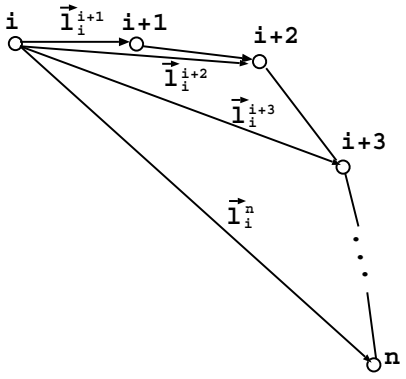


Figure 6.3: Accumulated torque due to distal segments

The accumulated torque acting on segment i is described by the equation

$$\vec{\tau}_i = \sum_{j=i+1}^{n} (\vec{l}_{i+1}^{\,j} \times \vec{F}_j), \tag{6.14}$$

where $\vec{l}_{i+1}^{\,j}$ is the vector from mass $i+1$ to mass j (note: this not necessarily a link) and F_j is the gravitational force acting on mass j. Instead of calculating all of these torques at each link, Jirasek derived an equivalent recursive formulation:

$$\vec{\tau}_i = \vec{l}_{i+1} \times (M_{i+1}\vec{g}) \; + \; \vec{\tau}_{i+1}. \tag{6.15}$$

In other words, an equivalent result can be obtained by adding the accumulated torque acting on the immediate child to the torque on the parent due to the total weight supported by the child [25]. Figure 6.4 helps to illustrate the derivation that follows.

Summation of vectors produces

$$\vec{l_i^{\,j}} = \vec{l_i^{\,i+1}} + \vec{l_{i+1}^{\,j}}, \tag{6.16}$$

which, when substituted into equation 6.14, gives

$$\vec{\tau}_i = \sum_{j=i+1}^{n} (\vec{l_i^{\,i+1}} + \vec{l_{i+1}^{\,j}}) \times \vec{F}_j. \tag{6.17}$$

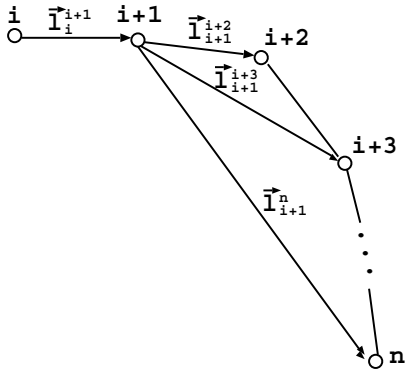


Figure 6.4: Accumulated torques at child segment

The cross-product is distributive, so

$$\vec{\tau}_i = \sum_{j=i+1}^{n} (\vec{l_i^{i+1}} \times \vec{F}_j) + \sum_{j=i+1}^{n} (\vec{l_{i+1}^{j}} \times \vec{F}_j), \tag{6.18}$$

$$= \sum_{j=i+1}^{n} (\vec{l_i^{i+1}} \times \vec{F}_j) + \vec{\tau}_{i+1}. \tag{6.19}$$

Using Equation 6.16, this becomes

$$\vec{\tau}_i = \vec{l_{i+1}} \times (M_{i+1}\,\vec{g}) \; + \; \vec{\tau}_{i+1}, \tag{6.20}$$

which is the desired result.

Equation 6.20 represents the accumulated torque acting on segment i. This torque causes a bend in the elastic rod by influencing the rotational velocity, $\vec{\Omega}$. As described above, $\vec{\Omega}$ is updated by the value of this accumulated torque divided by the rigidity of the rod. The flexural rigidity value $\Re_f$ is used to divide the $\vec{L}$ and $\vec{U}$ components of $\vec{\Omega}$, and the torsional rigidity value $\Re_t$ divides its $\vec{H}$ component,

$$\Omega_H^{new} = \frac{\tau_H}{\Re_t} \tag{6.21}$$

$$\Omega_L^{new} = \frac{\tau_L}{\Re_f} \tag{6.22}$$

$$\Omega_U^{new} = \frac{\tau_U}{\Re_f}. \tag{6.23}$$

Because the model uses a relaxation method, $\vec{\Omega}^{new}$ does not directly replace $\vec{\Omega}$. Rather, the model uses $\vec{\Omega}^{new}$ along with the rest rotation rate $\underline{\vec{\Omega}}$ to update the existing value, using the following update formula:

$$\vec{\Omega}' = (1 - \alpha)\vec{\Omega} + \alpha(\underline{\vec{\Omega}} + \vec{\Omega}^{new}). \tag{6.24}$$

Here, $\vec{\Omega}$ is updated to include a linear combination of its old value and a new value, the rest curvature $\underline{\vec{\Omega}}$ plus displacement caused by torque. α controls the speed of convergence to the new solution and may be reduced to prevent numerical instability.

The updated rotational velocity value $\vec{\Omega}'$ is used as $\vec{\Omega}$ in the next forward pass to reorient the child segment. This change in orientation affects the torque acting on the parent in the next backward pass, and the relaxation cycle continues. In Jirasek's model, convergence to equilibrium was assumed to occur after a certain number T of iterations. In contrast, I assume that equilibrium has been achieved when the maximum change in curvature in any segment in the model is below a user-defined threshold value.

Once equilibrium has been reached for the existing structure, a new *time step*, or step of growth, occurs. Primary growth is characterized by the addition of distal segments to the model.

6.1.5 Adding a segment

Addition of a new distal segment affects previous segments in several ways. First, the weight of the new segment increases the torque acting on all previous segments as in Equation 6.3. Second, all previous segments must grow *radially*, or in girth, according to the pipe model of Equation 2.2.

Radial growth of a stem occurs by adding rings, or layers of wood, around the existing rod [15]. These layers increase rigidity and also reinforce the current shape of the stem. An older, bulkier stem is therefore less likely to change its curvature even if there were a

sudden change in the external forces in the system. This phenomenon is called *branch shape memory* [14].

When new radial layers are added, the additional flexural rigidity of the ring is calculated using the moment of area for a hollow beam with circular cross-section,

$$I_{ring} = \pi \frac{r_{out}^4 - r_{in}^4}{4} \tag{6.25}$$

where r_{in} and r_{out} measure the distance from the center of the circle to the inner and outer edges of the ring, respectively [33].

The torsional rigidity of the ring is based on the polar moment of area for the same ring shape,

$$J_{ring} = \pi \frac{r_{out}^4 - r_{in}^4}{2}. \tag{6.26}$$

The flexural and torsional rigidities $\Re_{ring_f}$ and $\Re_{ring_t}$ of the ring are therefore

$$\Re_{ring_f} = E\, I_{ring} \tag{6.27}$$

$$\Re_{ring_t} = G\, J_{ring}. \tag{6.28}$$

To calculate the new rest rotational velocity $\underline{\vec{\Omega}}'$ for the thickened stem, the weighted average of the $\underline{\vec{\Omega}}$ values of the inner core and new outer ring is used [39]:

$$\underline{\Omega}'_H = \frac{\Re\, \underline{\Omega}_H + \Re_{ring_t}\, \underline{\Omega}_{ring_H}}{\Re + \Re_{ring_t}} \tag{6.29}$$

$$\underline{\Omega}'_L = \frac{\Re\, \underline{\Omega}_L + \Re_{ring_f}\, \underline{\Omega}_{ring_L}}{\Re + \Re_{ring_f}} \tag{6.30}$$

$$\underline{\Omega}'_U = \frac{\Re\, \underline{\Omega}_U + \Re_{ring_f}\, \underline{\Omega}_{ring_U}}{\Re + \Re_{ring_f}} \tag{6.31}$$

where the torsional rigidity value $\Re_{ring_t}$ is used to calculate the $\vec{H}$ component of $\underline{\vec{\Omega}}'$ and the flexural rigidity $\Re_{ring_f}$ is used for the $\vec{L}$ and $\vec{U}$ components of $\underline{\vec{\Omega}}'$, following the form of Equations 6.10 - 6.12.

Because new radial growth reinforces the current shape of the stem, the rest curvature of the outer ring $\underline{\vec{\Omega}}_{ring}$ is assigned so as to reflect the existing curvature of the stem:

$$\underline{\vec{\Omega}}_{ring} = \vec{\Omega}. \tag{6.32}$$

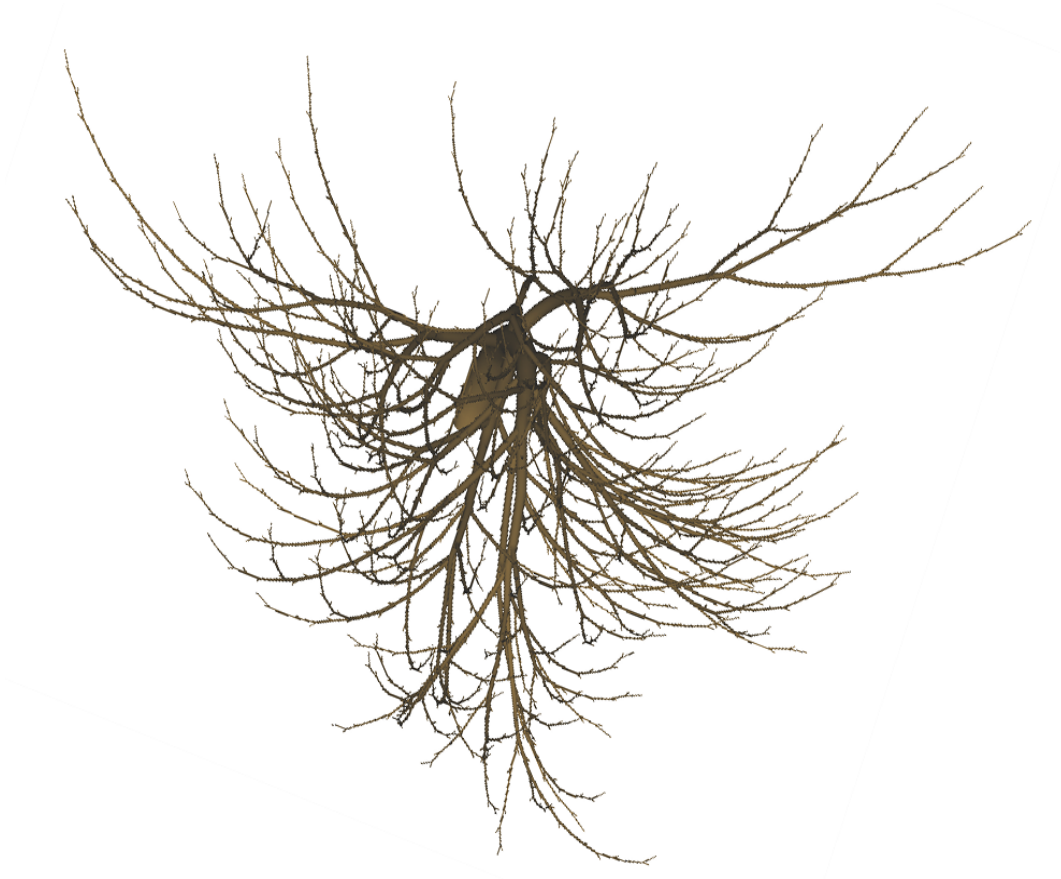


Figure 6.5: Lateral branching

Note that equation 6.32 is altered in Section 8.3 when reaction wood is simulated to actively reorient the stem.

The stem's new total rigidity becomes the combined rigidity of the inner core and outer layer,

$$\Re'_f = \Re_f + \Re_{ring_f} \tag{6.33}$$

$$\Re'_t = \Re_t + \Re_{ring_t}. \tag{6.34}$$

6.2 Lateral branching

The problem of representing a curved plant axis becomes even more interesting when one considers lateral branching. As depicted in Figure 6.5, lateral branching introduces non-

infinitesimal rotations at branching points due to divergence and deviation angles. Proper representation of biomechanics in the context of these non-infinitesimal rotations requires changes to Jirasek's implementation. This section provides a description of my extension to Jirasek's model which accurately represents biomechanics at branching points.

Lateral branching is accomplished by modifying frame orientations to account for divergence and deviation angles. This orientation change is achieved by spiralling around the $\vec{H}$ axis by the divergence angle ϕ and then rotating by the deviation angle θ. These rotations introduce a discontinuity in the curvature of the branch. Because these branching angles are non-infinitesimal rotations, they cannot be represented as vectors [1]. Instead, all branching rotations must be performed using quaternions.

The following pseudocode rotates a child's $\vec{H}\vec{L}\vec{U}$ frame by both the divergence and deviation angles required for branching:

$$\text{qDivergence} = \text{new Quaternion}(\vec{H}_{i-1}, \phi)$$

$$\vec{N} = \text{qDivergence}^{-1} * \vec{U}_{i-1} * \text{qDivergence}$$

$$\text{qDeviation} = \text{new Quaternion } (\vec{N}, \theta)$$

$$\text{qBranching} = (\text{qDivergence} * \text{qDeviation})$$

$$\vec{H}_i = \text{qBranching}^{-1} * \vec{H}_{i-1} * \text{qBranching}$$
$$\vec{L}_i = \text{qBranching}^{-1} * \vec{L}_{i-1} * \text{qBranching}$$
$$\vec{U}_i = \text{qBranching}^{-1} * \vec{U}_{i-1} * \text{qBranching}$$

At branching points, rotations due to biomechanics must still be considered. In Equation 6.1, it was assumed that only infinitesimal rotations were being performed, and therefore the vector cross-product was used to perform rotations [39]. However, at branching points, the simulation must compose these infinitesimal biomechanical rotations with the

above non-infinitesimal branching rotations. Therefore the rotation of Equation 6.1 must be reimplemented using quaternions.

Recall from Section 6.1 that the rotational velocity $\vec{\Omega}$ specifies a rotation per unit *length*. For each unit length of the parent segment, a rotation is performed around axis $\frac{\vec{\Omega}}{\|\vec{\Omega}\|}$ by angle $\|\vec{\Omega}\|$. Thus the total rotation along a parent segment of length $|\vec{l_{i-1}}|$ is of angle $\|\vec{\Omega}\| * |\vec{l_{i-1}}|$ around the same axis. This axis and angle are used to define a quaternion *qBending*.

$$\text{Quaternion qBending}(\frac{\vec{\Omega}_{i-1}}{\|\vec{\Omega}_{i-1}\|}, \quad \|\vec{\Omega}_{i-1}\| * |\vec{l_{i-1}}|)$$

Then, the reorientation due to the combined effects of branching and biomechanics becomes

$$\text{Quaternion qComplete} = (\text{qBranching} * \text{qBending})$$
$$\vec{H}_i = \text{qComplete}^{-1} * \vec{H}_{i-1} * \text{qComplete}$$
$$\vec{L}_i = \text{qComplete}^{-1} * \vec{L}_{i-1} * \text{qComplete}$$
$$\vec{U}_i = \text{qComplete}^{-1} * \vec{U}_{i-1} * \text{qComplete}$$

In the simulation's backward pass, a parent branch just before a bifurcation must accumulate the mass, girth, and torque information from both child branches. This is accomplished using the following formulae:

$$r_{i-1} = \sqrt[P]{r_i^{L^P} + r_i^{R^P}}, \tag{6.35}$$

$$M_{i-1} = m_{i-1} + M_i^L + M_i^R, \tag{6.36}$$

$$\vec{\tau}_{i-1} = \vec{\tau}_i^L + \vec{\tau}_i^R, \tag{6.37}$$

where m_{i-1} is the mass of segment $i-1$ itself and P is the pipe model constant, usually ranging between 2.49 for large trees to 3.0 for smaller trees [34]. Superscripts L and R represent left and right children.

6.3 Buckling tests

One relevant way of assessing the accuracy of the biomechanical model is to subject it to tests from elasticity theory. I have derived one such test from Euler's buckling equations, which determine when a long vertical column will bend under an axial load. The equations given in [35] have been modified to deal with buckling from a column's own weight only.

The length at which a vertical column will buckle under self-weight can be calculated from Equation 6.38. First, consider a perfectly vertical column of length l having uniformly distributed load q for each unit length. In this case, q is due to self-weight under gravity, so the incremental load q is

$$q = \rho \; a \; g, \tag{6.38}$$

where ρ is the density of the material, $g = 9.81 \frac{m}{s^2}$ is the magnitude of the gravity vector, and $a = \pi r^2$ is the area of the circular cross-section. Multiplication of this incremental load q by the length l of the column gives the total load ql. According to Euler, a column of this load and length will buckle if

$$ql >= ql_{cr} = \frac{7.84EI}{l^2}. \tag{6.39}$$

where E is Young's modulus of elasticity for the material (wood in this case) and I is the moment of area which equals $\frac{\pi r^4}{4}$ for a cylinder of circular cross-section [35].

Equation 6.39 can be used to determine the length l at which the column will buckle. The column must be perfectly straight and vertical, and be subjected to a small lateral force – a push, for instance. If the column's total load is less than ql_{cr}, it will regain its verticality as soon as the lateral force is removed. However, if the buckling length is exceeded, the column will remain bent when the lateral force is removed, and not return to vertical.

Although the column remains bent when the lateral force is removed, this form of buckling is still an elastic phenomenon. If the axial load (here, the column's self-weight) were to be somehow reduced (by cutting off its top, for instance), the deflected column would

straighten completely.

The above equation is accurate for an *ideal column* which has the following properties:

- the column is perfectly straight and has uniform cross-section throughout its length,
- the column is made from an isotropic material,
- the column is anchored at its based and free and its top,
- the stress due to loading cannot exceed the compression strength of the material [35].

Clearly, these criteria may not all be true of a tree branch or trunk. However, to test my method I modelled such an ideal column, having uniform cross-section and initially standing perfectly erect.

Buckling tests were simulated on this ideal column. Because the column model was made from many individual segments which were to act as one, each segment was given a small initial rotational velocity, simulating a small lateral push at the start of the simulation. I then observed whether the column buckled under this push. If it did not buckle, but returned to its vertical state, the length of the column was increased and the test repeated until elastic buckling occurred.

The buckling lengths thus found conformed to the equation above, within a margin of error of 0.1m. Until the value of ql exceeded the buckling limit ql_{cr} as defined above, the column righted itself. However, once $ql > ql_{cr}$, the column buckled. In the tests, segments of length 0.1m with radius 0.05m were used, and the length of the column was incremented by adding new segments. The buckling test was performed on ideal columns simulating wood of several species, using measured densities and Young's moduli for these woods obtained from [35]. Table 6.1 lists the values used and the resulting buckling lengths obtained from the tests. The lengths at which buckling occurred in the models are compared with the buckling lengths obtained from Equation 6.39.

Species	E	ρ	model buckling length	predicted buckling length	ql at predicted buckling length	ql_{cr} at predicted buckling length
Yellow Poplar	10.38	427	23.1	23.0	756.7	755.1
Apple	8.77	745	18.1	18.1	1038.9	1030.2
Black Willow	5.03	408	18.4	18.4	578.4	571.8
Black Oak	11.31	669	20.4	20.4	1051.5	1045.9
Yellow Birch	14.53	668	22.1	22.2	1142.6	1034.6
Silver Maple	6.22	506	18.3	18.4	717.3	707.0

Table 6.1: Table of buckling lengths for ideal columns made from various species of woods

Figure 6.6 compares the equilibrium states of the apple wood column at lengths 18.0m and 18.1m. At length 18.0m, buckling length had not been reached and the column was able to completely right itself. At length 18.1m, buckling occurred. This buckling length matches the length predicted by Euler's equation.

Figure 6.6: Equilibrium positions of simulated apple wood column at lengths (a) 18.0m and (b) 18.1m, after buckling occurred.

The tests produced buckling lengths which were within 0.1m of the lengths predicted by equation 6.39. The positive results of these tests validate the modelling approach in terms of its fidelity to the laws of elasticity theory for an ideal column.

6.4 Results and discussion

Figure 6.7 shows the "S" shape predicted by Fournier et al. [15] which results when the competing influences of torques due to gravity and negative geotropism act on a branch undergoing radial growth.

Figure 6.7: "S" shape of a stem due to self-weight and negative geotropism

The simulations described in this thesis provides significant improvements to efficiency when compared with the results of Jirasek et al. Although the running times for Jirasek's simulations were not recorded, they were estimated to take tens of minutes for simulations of single axes [37]. Animations representing growth could not be viewed interactively, but only by creating a video from captured frames. In contrast, the biomechanical simulation presented in this thesis achieves interactive rates until the trees become quite complex. Including reaction wood and breakage simulations yet to be described, the simulation runs at around 60 frames per second while the model contains less than 150 segments, above 30 frames per second until 800 segments, 20 frames per second until 1500 segments and 10 frames per second until 2500 segments. As such, interactive rates are maintained until the tree becomes very large: all images produced for this thesis consist of less than 2500 segments.

The importance of biomechanics in determining tree shape is made evident in Figure 6.8. Image (a) is a photograph providing the inspiration for the model. Image (b) shows the tree model, and (c) shows the same model but with the biomechanics removed.

Production rules for the biomechanical model expressed in LPFG are given in the Appendix.

(a)

(b) (c)

Figure 6.8: Tree model: (a) inspiration from nature (b) the model with biomechanics (c) the model without biomechanics.

Chapter 7

Case study: Crooked poplar tree

As a case study for the biomechanical simulation, I have modelled the "crooked" mutant of *Populus tremuloides*, found uniquely in Hafford, Saskatchewan. This population of trees has been the subject of several studies [43]. *Crooked poplar* trees have curved and twisted branches, which create a bush-like architecture, instead of a tall straight trunk (Figure 7.1). The crookedness of this mutant's branches is caused by severe bending of branches. When a bending branch becomes pendulous, one or more of its upward-growing lateral shoots take over as the dominant shoot. These so-called *relay shoots* usually occur at the middle of the parent branch, emerging from its highest point before it becomes pendulous [43]. The relay shoots grow quite long, claiming significant resources so that the pendulous portion of the parent branch loses vigour. This pendulous portion is rarely productive and is eventually shed. During the subsequent year, the vigorous relay shoots themselves bend in response to gravity, and pass dominance to their own lateral relay shoots. A detailed description of this mutant tree, including measurements, is given in [43].

The reasons for the increased bending in crooked poplar branches is unknown. There are at least two possible hypotheses. Bending could occur passively, as a result of reduced stiffness in the crooked mutant. Alternatively, bending could be active, so that branches undergo differential growth which forces them to grow downward.

In this simulation I have investigated the first hypothesis, that the mutant produces more supple wood than wild type poplar trees. Using the biomechanical model of Chapter 6 and personal communication from Dr. William Remphrey in the Department of Plant Science at the University of Manitoba, I have created a single model which can represent both the wild type and the crooked poplar tree.

Figure 7.1: Populus tremuloides, (a) wild type and (b) crooked mutant. Taken with permission from [43].

7.1 Review of the biomechanical simulation

The biomechanical simulation is described in Chapter 6. The tree grows by extending existing branches at their distal ends. After every step of primary or secondary growth, the equilibrium architecture of the model is determined. Rigidities and torques due to self-weight are calculated according to Equations 5.10 and 6.20. Then, rotational velocities $\vec{\Omega}$ are updated according to Equations 6.23 and 6.24. Orientations of segments are updated as a result of these rotational velocity values, and then torques are recalculated for the new orientations. This cycle continues until equilibrium has been reached for the current architecture, at which time a new growth step can occur.

7.2 Modelling the crooked poplar

Because the model is simulating passive bending, the most important characteristic of the crooked poplar model is its low wood stiffness values E and G. The suppleness of wood, especially juvenile wood, allows the dramatic bending of branches in the model. To prevent

the righting of these branches, the mechanism for reaction wood production is removed by setting $\alpha = 0$ in Equation 8.25. Further, the simulation of negative geotropism is adjusted so as to reduce its influence on internodes produced later in the summer season. Because new internodes are appended distally, this means that tropisms are not as effective on the pendulous portions of branches, as noted in [43].

7.2.1 Allocation of resources and fate of buds

A detailed description of the resource allocation strategy incorporated in this model is given in Chapter 4. In the case of the crooked poplar, buds at the geometric top (the highest point in space) of a bending parent shoot are most likely to become vigourous, particularly if the buds have an initial orientation close to global up. As illustrated in Figure 7.1(b), these buds are the most likely to become the next year's relay shoots. The tendency of these topmost buds to be most vigourous is further indicated by measurements in [43], depicted there in Figures 11(b) and 13.

Therefore, I created a heuristic measure of the geometric height along the parent shoot and the orientation of the bud, and used this measure to influence the distribution of resources to favour topmost buds. Equation 4.10 is modulated by the following factor:

$$\Phi = \left(\frac{y_{bud} - y_0}{y_{max} - y_0} \right) \left(\vec{H}_{bud} \cdot \text{up} \right) \tag{7.1}$$

where y_{bud}, y_0 and y_{max} are the vertical heights of the bud, the beginning of the parent branch and the highest point along the parent branch, respectively. y_{max} is calculated in the model by determining the largest y value for each branch in an acropetal pass, then propagating this maximum backward in a basipetal pass. Φ is used as a multiplier in Equation 4.10 to bias the amount of resources granted to the lateral child. Equation 7.1 was designed so that it could also be used to model acrotony in the wild-type poplar model. As such, the use of Equation 7.1 to influence resource distribution produces either the relay shoot phenomenon or a simulation of acrotony, depending on the shape of the parent branch. This allows the

appropriate resource allocation strategy to be an emergent property of a model which can produce either the crooked or wild type poplar tree.

7.2.2 Shedding of unproductive branches

Observations of the crooked poplar indicate that branches are shed after they have been unproductive for three years [42]. As described in Section 4.3.2, a branch in the model is considered unproductive when it no longer undergoes secondary growth. When a branch does not grow radially, it is flagged as unproductive. If it does not grow during the subsequent three years, it is shed.

Because the vigourous relay shoots claim so much of the parent branch's resources, the pendulous portion of the parent branch often becomes unproductive and hence is shed. Figure 7.2 shows a photograph of a parent branch whose pendulous portion has been unproductive and will soon be shed. Figure 7.3 shows an unproductive pendulous portion in the model.

Figure 7.2: Parent branch with two vigourous relay shoots. Beyond the relay shoots is the unproductive pendulous portion of the parent branch which will soon be shed. Taken with permission from [43].

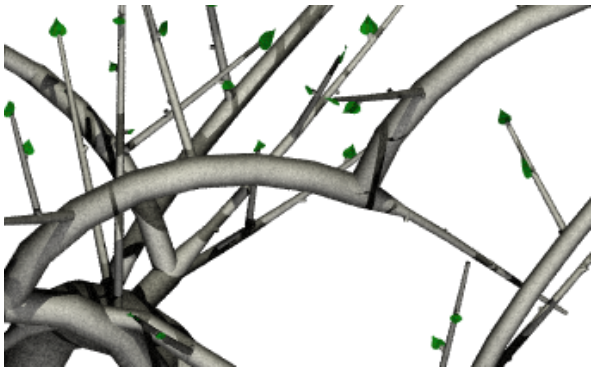


Figure 7.3: Unproductive pendulous portion of a parent branch in the model.

7.2.3 Other characteristics of the crooked poplar

Another distinguishing feature of the crooked mutant is its unusually large divergence angles as compared to the wild type poplar tree [43]. As seen in Figure 7.2, divergence angles in the crooked poplar can often be close to 90 degrees, whereas the wild type divergence angles average 56.23 degrees [43].

Measurements in [43] indicate that parent branches typically have between one and three vigourous relay shoots. More recently, it has been noted that the tendency of these shoots to become pendulous or remain upright can vary significantly, as seen in Figure 7.4. This variability is modelled by using a random number drawn from a normal distribution when assigning a juvenile stiffness value to each new shoot.

7.3 From wild type to crooked mutant

The model described in this chapter can simulate both the wild-type *Populus tremuloides* and the crooked mutant, allowing conversion between them by changing only a small number of parameters. Because the main parameter changed in this conversion is wood stiffness, the model demonstrates one possible cause for the mutant's crooked architecture – that of passive bending due to supple wood. Changes in a small number of the biomechanical model's parameters (listed below) results in a crooked tree, and as such the model provides

Figure 7.4: A crooked poplar tree displaying multiple upright shoots. Taken with permission from [42].

some justification for the theory of passive bending.

In summary, the changes to the model involved in switching from wild type to crooked type are as follows:

- reduce both juvenile and adult E by a factor of three,
- reduce the intensity of negative geotropism accordingly
- decrease the effect of negative geotropism during summer to simulate a shoot "giving up" and becoming pendulous,
- remove the mechanism for reaction wood production, and
- double the average deviation angle.

The most significant change to the model is the reduction of stiffnesses E and G. This is the main contributor to the crooked morphology of the mutant model, and accounts for the

curvature of its branches. Besides the changes listed above, all other differences in the two tree architectures occur as emergent properties of the model.

7.4 Results and discussion

Crooked and wild type poplar models may be compared in Figures 7.5 and 7.6.

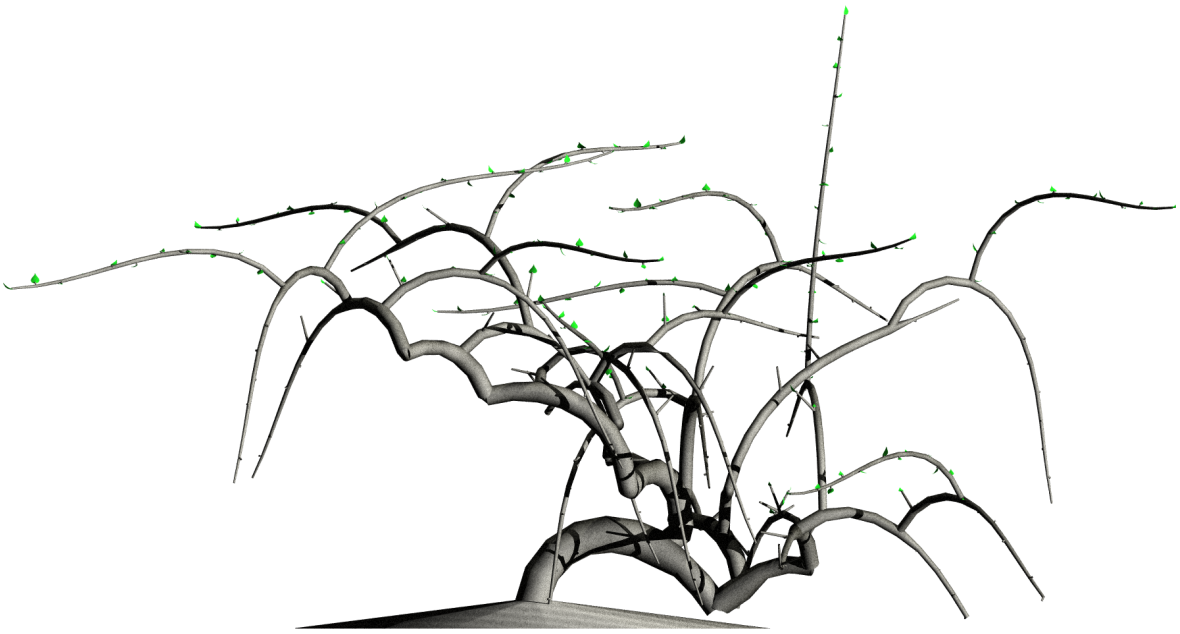


Figure 7.5: Crooked poplar model.

The most visually pleasing models have fewer relay branches per shoot on average than measurements indicated in [43]. This reduction in average number of relay shoots is by no means a requirement of the model, but the presence of large numbers of branches makes the resulting architecture more difficult to interpret visually.

Tests have not yet been made on the crooked poplar to determine its wood's stiffness properties. As such, this model is an investigation of one possible reason for the mutant's crooked architecture and not a recreation from measured stiffness data. Though the model produces convincing results, it does not constitute proof of the mutant tree's reduced wood stiffness.

Figure 7.6: Wild type poplar model.

Figure 7.7 shows the crooked poplar model from a different angle. Changes to the general biomechanical model's production rules, which allow for the representation of both crooked and wild-type poplar trees, are presented in the Appendix.

Figure 7.7: Crooked poplar model.

Chapter 8

Wood properties, reaction wood and breakage

In this chapter, further enhancements to the biomechanical model's biological realism are described. Elasticity theory concepts from Chapter 5 are applied to the model in combination with biomechanical findings relating specifically to wood and tree growth. Also described in this chapter is a simulation of wood's active reorientation due to reaction wood, and of branch breakage due to high bending stress.

Tree branches are subject to static loading from self-weight, as well as dynamic loading from other instantaneous forces. These lateral loads cause bending (and possibly breakage, to be discussed in Section 8.6). Often, a tree's growth is optimized so as to resist, or even correct, excessive bending.

As described in Section 5.4.1, there are two ways to increase a stem's resistance to bending. First, the stem can be made from a stiffer material, as evidenced by a higher Young's modulus E. Secondly, it can be composed of more material, so as to increase the moment of area I. Either approach will increase the stem's rigidity $\mathfrak{R}$, according to Equation 5.10, $\mathfrak{R} = E\,I$. In the following subsections, methods of increasing I and E are discussed.

8.1 Increasing the moment of area, I

Increasing girth is an ideal way of strengthening support and resistance to bending. An increase in the cross-sectional radius increases the area over which bending stresses are distributed. In trees, this increase in girth arises from the production of new secondary growth rings. The cambium, located just under the bark and phloem at the outer edge of the cross-section, is responsible for producing new wood cells which form these rings [53].

The impact on rigidity of adding a radial ring was described in Section 6.1.5. The rigidity

of the ring $\mathfrak{R}_{ring}$ is added to the rigidity of the existing branch $\mathfrak{R}$ to obtain the total rigidity $\mathfrak{R}'$ of the thickened branch.

Thus far, all branches and rings have been assumed to be circular in cross-section. Section 8.3 will examine ways in which the ring's cross-sectional shape can change in order to better resist a particular direction of bending.

8.2 Increasing Young's modulus, E

Wood is not homogeneous; it is composed of various compounds with different material properties. The stiffness of wood is due to its two component stiffening agents, cellulose and lignin.

The cell walls of wood are made mostly of cellulose (43 percent), hemicellulose (28-30 percent), and lignin (7 percent) [31]. Being the largest component of cell wall structure, cellulose is the basis for the overall properties of wood [27]. Cellulose is formed into *microfibrils*, or strands of molecules, oriented along the longitudinal axis of a stem. It has high tensile strength [32].

In contrast, lignin has high compressive strength [32]. Although lignin makes up a relatively small proportion of the cell wall structure, larger quantities of lignin are found in the *middle lamella* region between cells [31]. Thus it still contributes greatly to the stiffness of wood.

The combination of tensile and compressive strengths from cellulose and lignin produces a very strong composite material. As cited in [31], Freudenberg likened this combination of strengths in wood to that found in reinforced concrete:

> *The micelle series of cellulose may be compared with the iron rods, and the lignin together with the hemicellulose with the concrete, in reinforced concrete. It is astonishing to see how nature has here made use of two of the best principles*

of rigidity which the human mind has independently discovered only in our own time (cited in [31], page 18).

Because wood is a composite material, a value of E for one particular material cannot describe wood's stiffness. Instead, the *structural Young's modulus* for wood is calculated as the result of bending tests. Once rigidity $\Re$ is measured, Young's modulus can be determined using Equation 5.10 [7]:

$$E_{structural} = \frac{\text{observed flexural rigidity } \Re}{I}. \tag{8.1}$$

As well as being heterogeneous, wood is also *anisotropic*, having different strength and stiffness when tested in different directions [35]. Wood's anisotropy results from the alignment of its component substances [31]. As cellulose microfibrils are directed longitudinally, the measurement of stiffness E along the wood grain is significantly higher than its stiffness the other directions [31].

However, to simplify calculations I assume that wood is isotropic, and thus that E is constant in any direction of bending. (Note that this does not include torsion, for which wood's shear modulus G is used.) Mattheck [32] also made this simplification, claiming that the main flow of forces is directed along the grain of the wood .

8.2.1 Wood aging

As an approximation, one can assume that all wood exhibits the same mechanical properties, and use one value of Young's modulus E uniformly in the model. However, a more realistic simulation accounts for the stiffening of wood as it matures. Fournier described the younger *sapwood*, which forms new rings under the bark, as being more prone to bending than the *heartwood* at the centre of an older branch [14]. This behaviour of *juvenile* wood as compared to *mature* wood is accomplished in nature by one of three approaches:

1. higher density, same E;

2. same density, lower E; or

3. higher density, lower E,

where density is defined as volume divided by mass.

I chose to implement approach 2 in the simulation, keeping density constant and reducing the E value used for juvenile wood. When a new shoot is formed, the juvenile wood stiffness E_{juv} is used in rigidity Equation 5.10. As internal rings age, they undergo a gradual stiffening as Young's modulus approaches E_{adult}. At every step of simulation time, the preexisting wood's E value is incremented based on the branch's increase in age:

$$E'_{core} = E + (\Delta age * \frac{E_{adult} - E_{juv}}{age_{adult}}) \tag{8.2}$$

where age_{adult} is the age at which wood is fully mature and stops stiffening.

The overall stiffness of the branch segment is determined by the combined stiffnesses of all radial layers. Every time a new ring is added, its stiffness contributes to that of the entire branch. Using the stiffness values and moments of area for this ring and the preexisting core of the branch, the new rigidity $\Re'$ of the entire branch is determined, as in Equations 6.33 and 6.34:

$$\Re'_f = E_{core}\, I_{core} + E_{ring}\, I_{ring} \tag{8.3}$$

$$\Re'_t = G_{core}\, J_{core} + G_{ring}\, J_{ring}. \tag{8.4}$$

The stiffness of the entire segment can be obtained by dividing this rigidity by the new moment of area for the thickened segment:

$$E' = \frac{\Re'_f}{I_{circle}} \tag{8.5}$$

$$G' = \frac{\Re'_t}{J_{circle}}. \tag{8.6}$$

Equations 8.5 and 8.6 define the structural Young's modulus and shear modulus of the entire segment, to be used as future values of E_{core} and G_{core} when the next radial ring is added.

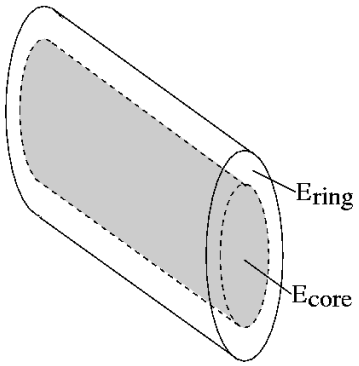


Figure 8.1: Branch segment with inner core of stiffness E_{core} and new outer ring of stiffness E_{ring}.

As described above, branch segments in the model stiffen because of two factors: increase in girth from the addition of radial rings, and stiffening due to aging of the segment's heartwood. Both of these effects reinforce the current shape of the branch, represented by $\vec{\Omega}$ in the model. This implies a smaller range of motion in response to applied forces. This phenomenon, described in Chapter 6, is called branch shape memory [15].

8.3 Reaction wood in nature

Any lean in a tree branch will intensify if new primary growth increases the torque it is experiencing. To control the lean, trees could discourage the development of new branches in locations which increase the torque felt by the leaning branch. This control over primary growth is an approach taken by Hart et al. to produce a balanced tree [21].

In contrast, Fournier argued that this redirection of primary growth is not sufficient to correct a lean. The portions of a tree undergoing primary growth account for only 15 to 30 percent of a tree's weight, so a leaning trunk cannot be corrected by primary growth alone [15]. Yet, straightening of leaning trunks is indeed observed, as shown in Figure 8.2. This is in part due to *reaction wood*, or differentiated wood produced by real trees to reorient a branch after deflection.

Reaction wood actively reorients an inclined branch by differentiating wood production

Figure 8.2: Photograph of a tree in which reaction wood has corrected a leaning trunk.

in the secondary growth. For simulation purposes, three phenomena evident in reaction wood must be considered. First, reaction wood produces *angular dissymmetry* in the cross-section by altering the growth speed in certain regions. This alters the shape of the cross-section, and as a result changes the I and J factors in the rigidity Equations 5.10 and 5.13. Secondly, it alters the material properties of the new wood, changing the E and G values in these equations. Finally, it introduces new growth strains which are directed against the lean. All of these changes allow the branch to better resist, and even correct, bending.

8.3.1 Detection of a lean

Attempts have been made to identify the trigger of reaction-wood production. Studies have shown that reaction wood forms when a growing shoot deviates too far from an *equilibrium position*, EP. The existence of such an EP is a widely held belief [54, 35, 30], although it is unclear how this position is determined mechanistically [35]. The three properties of reaction

wood described in the following sections act to reorient the branch back to its EP [35].

8.3.2 Differentiated shape

Reaction wood grows faster than regular wood, forming thicker portions of new rings on the *adaxial*, or upper, surface of a downward-bending branch in angiosperms, and on the *abaxial*, or underside, in conifers [33]. By thickening the appropriate angular section of the branch's new rings, reaction wood ovalizes the branch cross-section, as seen in Figure 8.3. This localized accelerated growth causes a dissymmetry which shifts the geometric centre of the branch (and therefore the zero-stress zone) according to Equation 5.15. It also increases the area over which the load is applied, thus reducing the stress by definition.

Figure 8.3: Cross-section showing asymmetrical thickening of rings due to reaction wood. The resulting elliptical cross-section aids to resist bending along the axis of thickening. Taken with permission from [17].

Because the cross-section is no longer circular, the moment of area I also changes. In the direction of bending, the branch has thickened and so I has increased; thus the branch is better able to resist further bending in this direction. In other directions, however, the branch has not thickened and so I has not increased. Material has not been wasted in thickening the branch in directions not currently subject to bending. Changes to the moment of area I are described in more detail in Section 8.4.2.

8.3.3 Differentiated material properties

As well as growing more quickly, reaction wood also exhibits different mechanical properties than regular wood. It is optimized either for compression or for tension, depending on which side of the stem it forms.

In angiosperms, reaction wood forms on the upper sides of stems and is optimized to resist tension. This *tension wood* acts by contracting, so as to resist the expansion of the wood on the top of the downward-bending branch [35]. Tension wood is characterized by gelatinous fibers in an inner cell wall layer which is almost without lignin [27]. As such it contains much of the cellulose which provides tensile strength, and little of the lignin which is strong under compression [32].

In conifers, reaction wood forms on the undersides of stems, and is optimized to resist compression [33]. This *compression wood* acts by expanding to resist the compression caused by the downward bending of the branch [35]. Compression wood cells are round and have large spaces between cells, whereas normal wood cells are closely packed polygons [27]. Since lignin lies in this enlarged region between cells, such wood has a much higher lignin content and is therefore strong under compression [27]. Compression wood has a comparatively lower cellulose content, which explains its low tensile strength.

8.3.4 Growth strains

The above changes to wood production cause *growth strains* in the stem. Reaction wood forms where the stress due to bending is highest [33], and the differential growth produces *growth stresses* to oppose these loading stresses [53]. The growth stresses lead to *growth strains* which counteract the existing strains due to loading [7]. These growth strains supply the moment needed to correct a lean [35].

In tension wood, the wood's contraction causes internal compressive growth stresses, which oppose the existing tensile stresses on the top of the branch. In compression wood,

expansion creates tensile growth stresses to oppose the compression of the underside of the branch. Both types of reaction wood act to correct a lean by opposing and attempting to neutralize the bending stresses felt in the reaction wood region.

8.4 Reaction wood simulation

The approach to reaction-wood modelling taken in this thesis is based on this equilibrium-position theory described in Section 8.3.1, and simulates the three characteristics of reaction wood described above.

8.4.1 Detection of a lean

Reaction wood is simulated with several steps. First, a lean is detected when a segment's actual rotational velocity $\vec{\Omega}$ deviates sufficiently from its EP, as represented by the segment's rest rotational velocity in the absence of loading, $\underline{\vec{\Omega}}$. If the deviation exceeds a user-controllable threshold ξ, the production of reaction wood is triggered.

$$\text{EP} = \underline{\vec{\Omega}} \tag{8.7}$$

$$\text{deviation from EP} = \underline{\vec{\Omega}} - \vec{\Omega} \tag{8.8}$$

As in real trees, all changes to wood production to correct a lean must occur in new wood rings formed after the detection of the lean. Therefore the preexisting core of the branch does not change with the production of reaction wood; only newly forming rings exhibit different properties when reaction wood is triggered.

8.4.2 Differentiated shape

As discussed above, reaction wood ovalizes the cross-section of the leaning branch. A model which allows for reaction wood must store lengths for both semi-axes of the cross-section's

ellipse, *a* and *b*. The larger semi-axis of the ellipse, *a*, lies in the direction of the lean.

Because bending in the model only occurs due to gravity, I assume that reaction wood forms along the axis of the cross-section closest to vertical, as in Figure 8.4. Each segment in the model keeps an orthogonal reference frame $\vec{H}\vec{L}\vec{U}$ which rotates so as to keep the $\vec{U}$ axis as close as possible to vertical given the heading vector, and therefore in line with the cross-section's longer semi-axis *a*. After every reorientation of the segment, the frame is rotated about its $\vec{H}$ axis in the following way:

$$\vec{U} = \vec{H} \times (\ (0,1,0) \times \vec{H}\ )$$
$$\vec{L} = \vec{U} \times \vec{H}$$

Given the heading vector $\vec{H}$, this method orients $\vec{U}$ as closely as possible to the vertical without compromising orthogonality of the reference frame, as in Figure 8.4.

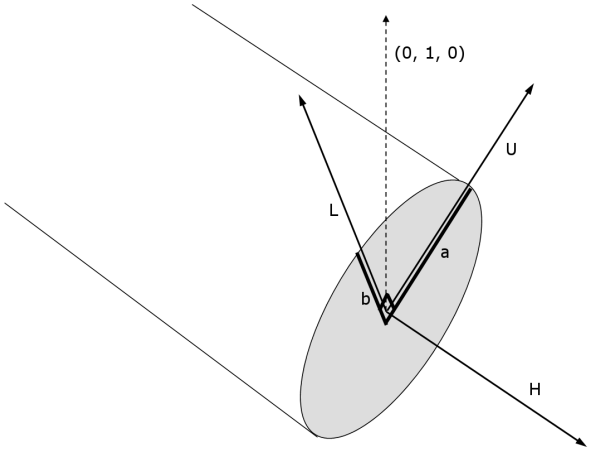


Figure 8.4: Elliptical cross-section with semi-axes *a* and *b*, showing the model's three orthogonal axes and the vector (0, 1, 0).

Because the cross-section is no longer circular, the moments of area now vary about each axis of the segment. Rotation about the frame's $\vec{L}$ axis creates a bend in the direction of the current lean which will be resisted by increased girth in this direction [33]:

$$I_{ellipse_{LL}} = \frac{\pi}{4} a^3 b. \tag{8.9}$$

Because $I_{ellipse_{LL}}$ depends so heavily on the longer semi-axis a, its value is large and therefore the branch is better able to resist bending in the direction of the lean. Rotation about the frame's $\vec{U}$ axis results in a bend along the cross-section's narrower axis. $I_{ellipse_{UU}}$ is therefore smaller as resistance to bending relies more heavily on the smaller semi-axis b:

$$I_{ellipse_{UU}} = \frac{\pi}{4} a b^3. \tag{8.10}$$

The torsional constant J also changes to reflect the elliptical cross-section. The value of J for an elliptical cross-section is [35]:

$$J_{ellipse} = \frac{\pi a^3 b^3}{a^2 + b^2}. \tag{8.11}$$

Note that if $a = b$, all moments of area equal those of a circular cross-section, $I_{circle} = \frac{\pi}{4} r^4$ and $J_{circle} = \frac{\pi}{2} r^4$. Thus the ellipse equations may be used even for branches of circular cross-section which are not producing reaction wood.

Whenever a new radial ring is produced, the deviation from EP is determined as in Equation 8.8. If the lean is too small for reaction wood to be triggered, a ring of constant thickness is formed. Otherwise, the ring grows thicker in the direction of bending.

In keeping with the pipe model theory,the girth of the parent cross-section at a bifurcation point is determined from the girths of both children. When allowing for reaction wood, however, this calculation must be performed using the cross-sectional area of child segments, instead of simply using radii as in Equation 2.2, because one or both of the children may have elliptical cross-sections which require two semi-axes instead of a single radius.

When using area instead of radius in the pipe model computation, the exponent P from the pipe model must be divided by two. The reason for this is made clear by considering the case of circular cross-sections, in which area depends on the square of the radius. For now,

let N be the exponent:

$$area_{parent} = \sqrt[N]{{area_1}^N + {area_2}^N} \tag{8.12}$$

$$\pi r^2_{parent} = \sqrt[N]{(\pi\ {r_1}^2)^N + (\pi\ {r_2}^2)^N} \tag{8.13}$$

$$\pi r^2_{parent} = \sqrt[N]{\pi^N\ \ ({r_1}^{2N} + {r_2}^{2N})} \tag{8.14}$$

$$r^2_{parent} = \sqrt[N]{{r_1}^{2N} + {r_2}^{2N}} \tag{8.15}$$

$$r_{parent} = \sqrt[2N]{{r_1}^{2N} + {r_2}^{2N}}. \tag{8.16}$$

Murray's observations of cross-section radius used in Equation 2.2 necessitate that $2N = P$. Thus, an exponent of $N = \frac{P}{2}$ is used when performing the pipe model computation using areas instead of radii.

$$area_{parent} = \sqrt[\frac{P}{2}]{{area_{child1}}^{\frac{P}{2}} + {area_{child2}}^{\frac{P}{2}}} \tag{8.17}$$

Because a parent segment may itself have accumulated reaction wood previously, its cross-section may not be circular. Therefore, once the area $area_{parent}$ is determined from Equation 2.2, it must be used to calculate lengths of ellipse semi-axes a and b:

$$area_{parent} = \pi\ a\ b \tag{8.18}$$

One more constraint is needed to solve Equation 8.18 for the two unknowns a and b. This constraint should maintain the existing *shape* of the ellipse, in other words, the ratio between the lengths of the semi-axes. If the parent segment has never developed reaction wood, it will remain circular and have $a = b$. If it has previously developed reaction wood, its current elliptical proportions will be maintained even as its cross-sectional area increases according to Equation 8.18. This is achieved by maintaining the existing ratio ζ between the lengths of the ellipse's semi-axes:

$$b = \zeta\ a \tag{8.19}$$

Equations 8.18 and 8.19 form the system of equations needed to solve for the new semi-axis lengths a and b.

If reaction wood production has been triggered, the long semi-axis a is further lengthened after the above calculations are performed:

$$a \mathrel{+}= \text{ reaction wood increment} \tag{8.20}$$

The new reaction wood rings will be more elliptical than the preexisting inner core, which corresponds to a ring cross-section like that of Figure 8.5.

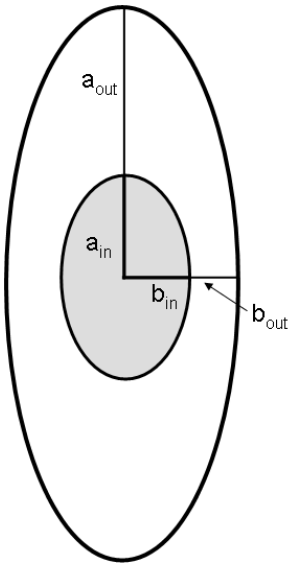


Figure 8.5: Preexisting inner core with new elliptical reaction wood ring

The moments of area I about the $\vec{L}$ and $\vec{U}$ axes for such a ring are

$$I_{ring_{LL}} = \frac{\pi}{4}(a_{out}^3 b_{out} - a_{in}^3 b_{in}) \tag{8.21}$$

$$I_{ring_{UU}} = \frac{\pi}{4}(a_{out} b_{out}^3 - a_{in} b_{in}^3). \tag{8.22}$$

As explained above for $I_{ellipse}$, these equations are valid for all radial rings, whether or not reaction wood is present. They are also valid whether or not the preexisting inner core was itself elliptical. However, for simplicity the longer semi-axis of the inner core a_{in} is assumed to lie in the same direction as a_{out}, as in Figure 8.5. In other words, the axis of bending in the branch is assumed to remain constant while reaction wood is being produced.

8.4.3 Differentiated material properties

The optimization of reaction wood's material properties is simulated by using a larger Young's modulus E in reaction wood than that which is used for regular secondary growth. However, because the model does not consider angular portions of the cross-section separately, different E values cannot be assigned to the top and bottom of a stem. Therefore a segment's overall stiffness is computed as the average of its reaction wood (E_{rw}) and non-reaction wood (E_{juv}) components, estimating from Figure 8.3 that about half of the new ring's wood is reaction wood.

$$E_{ring} = \frac{1}{2}E_{rw} + \frac{1}{2}E_{juv} \tag{8.23}$$

Because the angular portion of the cross-section that contains reaction wood is not identified explicitly, both angiosperms and conifers can be modelled using the same technique.

This E_{ring} value is used to calculate the rigidity of the ring $\mathfrak{R}_{ring_f}$, according to Equation 6.27. The rigidity of the ring $\mathfrak{R}_{ring}$ is added to that of the inner core according to Equation 6.33, to obtain the flexural rigidity of the entire segment. An equivalent calculation is made for torsional rigidity.

8.4.4 Growth strains

Alterations to the shape and stiffness of reaction wood can slow or even stop bending, but they cannot actually reverse the direction of bending to correct a lean. The third aspect of the reaction wood model simulates the growth strains which, in nature, provide a rotational moment for righting a branch. This is done by adjusting the rest curvature $\underline{\vec{\Omega}}_{ring}$ of the new ring in the direction opposite to the lean.

The $\underline{\vec{\Omega}}$ value of the inner core of the branch describes the curvature it would attain in the absence of load (Section 6.1). The load causes the branch to lean away from its desired curvature $\underline{\vec{\Omega}}$ to an actual curvature $\vec{\Omega}$. Usually, a newly created secondary growth ring uses

the actual curvature $\vec{\Omega}$ of the inner core as its rest curvature, as in Equation 6.32:

$$\underline{\vec{\Omega}}_{ring} = \vec{\Omega}. \tag{8.24}$$

This reinforces the current shape of the branch, thereby accomplishing the simulation of branch shape memory as described in Section 5.4.

However, when reaction wood is present, branch shape memory is not accomplished. Instead of reinforcing the current shape of the branch, the new reaction wood ring acts to pull or push the branch away from the direction of the lean. To accomplish this, the rest rotational velocity $\underline{\vec{\Omega}}_{ring}$ of the new ring is adjusted in the direction of $\underline{\vec{\Omega}}$, which is the segment's EP (Equation 8.7):

$$\underline{\vec{\Omega}}_{ring} \; = \; \vec{\Omega}(1-\beta) \; + \; \underline{\vec{\Omega}}\beta. \tag{8.25}$$

The higher the proportion of reaction wood β set for the model, the more strongly $\underline{\vec{\Omega}}$ affects the rest curvature of the reaction wood ring. Equation 8.25 replaces Equation 6.32 for $\underline{\vec{\Omega}}_{ring}$ in Jirasek's model. Note that if the user has specified no reaction wood in the model ($\beta = 0$), the result is the same as that of Equation 6.32.

When the rest rotational velocity $\underline{\vec{\Omega}}'$ of the thickened segment is calculated using the weighted average of $\underline{\vec{\Omega}}$ and $\underline{\vec{\Omega}}_{ring}$ as in Equations 6.29 to 6.31, a reorientation of the thickened segment in the direction away from the lean is accomplished. Because ring rigidity $\Re_{ring}$ is large in reaction wood, the influence of the new ring on $\underline{\vec{\Omega}}'$ is significant (see Equations 6.29 to 6.31).

Equation 8.25 is similar to Equation 6.24, which describes the influence of tropisms. The growth strain aspect of reaction wood behaviour is implemented similarly to tropisms in the model, except that it alters the rest rotational velocity value in the absence of load, $\underline{\vec{\Omega}}$, instead of the existing rotational velocity $\vec{\Omega}$. Also, growth strains caused by reaction wood affect only newly created rings on preexisting segments, whereas tropisms influence newly created segments.

8.5 Reaction wood results and discussion

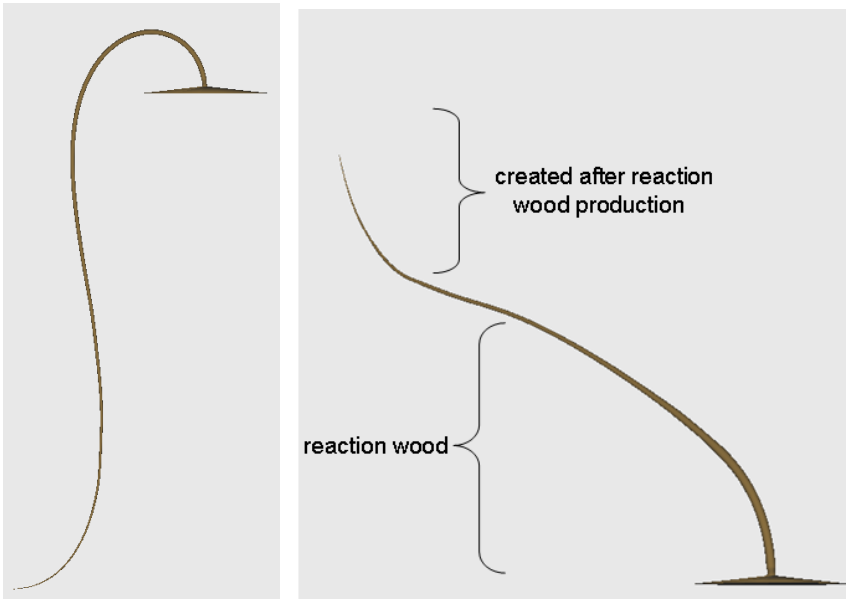


Figure 8.6: A single stem (a) without reaction wood (b) with reaction wood.

Because reaction wood occurs only in new radial rings produced after the detection of a lean, no reaction wood response occurs until new radial growth is produced. No growth occurs in the model until equilibrium has been reached for the existing structure. Therefore, a lean will not be corrected by reaction wood production until the process of leaning is complete. However, radial rings containing reaction wood are produced before further distal segments can be added to the branch. Therefore, reaction wood response occurs before the increased torque from new segments can augment the lean.

Figure 8.6 shows the same leaning stem with and without reaction wood in the model. In the absence of reaction wood production, the stem achieves the "S" shape predicted by Fournier et al. [15] (Figure 8.6(a)). With reaction wood, the lean is resisted and new growth resumes an upward direction (Figure 8.6(b)).

Figure 8.7 shows a tree before and after reaction wood production. Note that the lean in the leftmost lateral branch as well as the trunk has been corrected. The increase in trunk girth dictated by the pipe model results from reaction wood production in the branches.

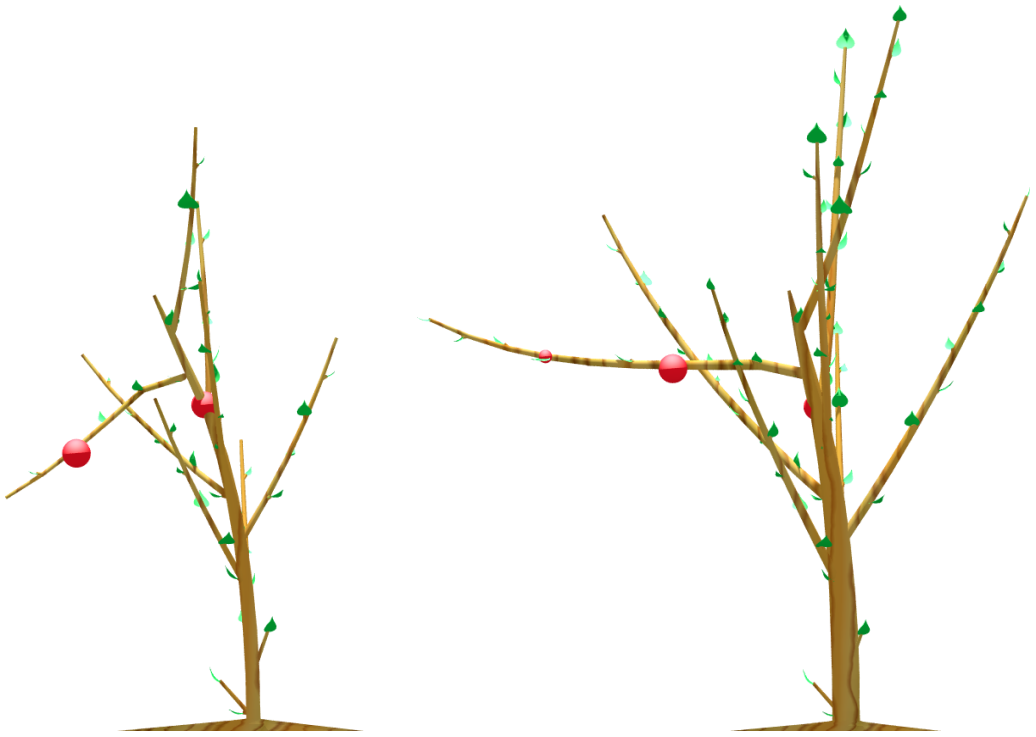


Figure 8.7: A simple tree, before and after reaction wood is produced.

8.6 Breakage

The use of concepts from elasticity theory, realistic material properties and biomechanics calculations allows the model to handle testing for branch breakage. Fracture of tree branches usually begins with failures due to normal (tensile or compressive) stresses. Any failures due to sheer stresses occur only afterwards [35]. For this reason, sheer stresses due to bending can be ignored when determining failure.

To determine whether an object will break, the largest bending stress experienced in the cross-section is compared to the breaking strength of the material. As defined in Section 5.5, bending stress is:

$$\sigma_B = \frac{M\underline{r}}{I}, \tag{8.26}$$

where M is the bending moment, $\underline{r}$ is the distance from the centroid axis, and I is the moment of area for the segment. Recall that σ_B is a compression stress on the underside of a bending

branch and a tensile stress on the upper side.

When the stress due to bending σ_B is below the material's strength, the branch does not break. Instead, its deformation causes storage of strain energy, which is converted to kinetic energy and used to restore the original shape after the load is removed [35]. In the model, this strain is represented by the difference between the desired, or rest, rotational velocity $\underline{\vec{\Omega}}$ and the actual value $\vec{\Omega}$.

In contrast, if the stress σ_B exceeds the breaking strength, this strain energy is instead used to cause breakage by propagating a crack. Before a crack initiates, wood typically undergoes a compression failure. Because wood is weaker under compression, the first point of failure occurs on the underside of the branch by the formation of a kink, when individual fibres buckle. This compression failure is followed by a crack on the upper side which is under tension [33]. The stored strain energy is then used to propagate the crack across the entire cross-section [35].

The *modulus of rupture*, *MOR*, defines the stress required for fracture. Thus, breakage occurs if

$$\sigma_B > MOR, \tag{8.27}$$

where σ_B is computed as in Equation 5.15. For example, $MOR = 64$ MPa for poplar wood [35].

8.7 Simulation of breakage in a fruit tree

In most tree species, breakage due only to stresses from self-weight is unlikely. Trees are often overbuilt in terms of support for static loads such as self-weight. However, in some fruit trees branches do break because of the weight of large fruit. A simulation of a heavy-laden fruit tree is used to illustrate breakage in the model.

At every simulation step, the maximum stress experienced by each branch segment is computed, and breakage tests are performed according to Equation 8.27. If a segment's

maximum stress exceeds the modulus of rupture, both the segment and its supported subtree are removed from the tree. As a result of the breakage of a branch, the rest of the tree springs up in response to the reduced load, as shown in Figure 8.8.

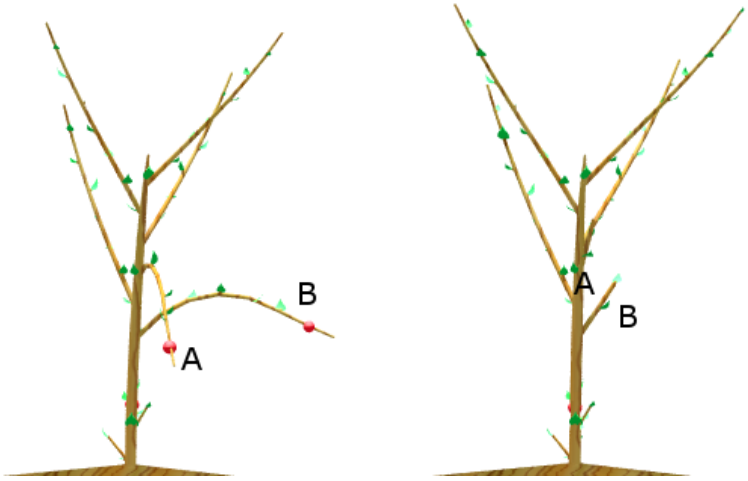


Figure 8.8: A simple tree, before and after the breakage of two branches A and B due to stress from weight of fruit.

Stress due to bending σ_B is at a maximum at the base of a branch, because this is where the torque lever arm is longest [35]. This is why many trees develop *branch collars* at the base of large branches, to increase I by thickening this most-stressed portion of the branch [35]. In the biomechanical model, a segment at the base of a branch may have been built up radially before fruit produced distally created a large torque. In this case, the stress at this segment is reduced according to Equation 5.15 and breakage is more likely to occur at a thinner segment further from the branching point, as shown in Figure 8.8(b).

8.8 Breakage and reaction wood

Heavily leaning branches such as A and B in Figure 8.8 are likely to either break or produce reaction wood, because both phenomena are triggered by related occurrences. Reaction wood is produced if the bend exceeds the threshold ξ, and breakage occurs when stress

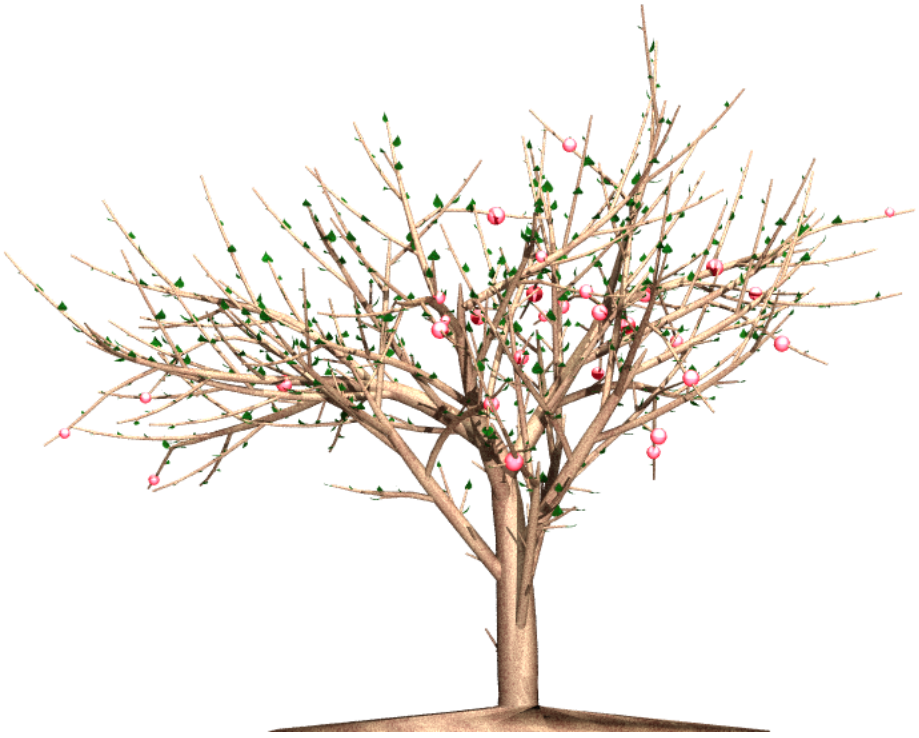
exceeds the threshold *MOR*. Both large bends and large stresses are caused by a large torque, such as is caused when a heavy fruit is produced on a slender new branch.

The occurrence of either reaction wood or breakage preempts the occurrence of the other. Which phenomenon occurs first depends on the thresholds ξ and *MOR*. Often breakage occurs immediately after a new shoot begins to produce a heavy fruit, because the stress due to weight on the fragile new segment exceeds the breaking strength. If this stress exceeds the *MOR* before any new radial rings can be added, then reaction wood can have no effect, as it is produced only in new radial rings. In this scenario, the branch breaks before reaction wood can be produced.

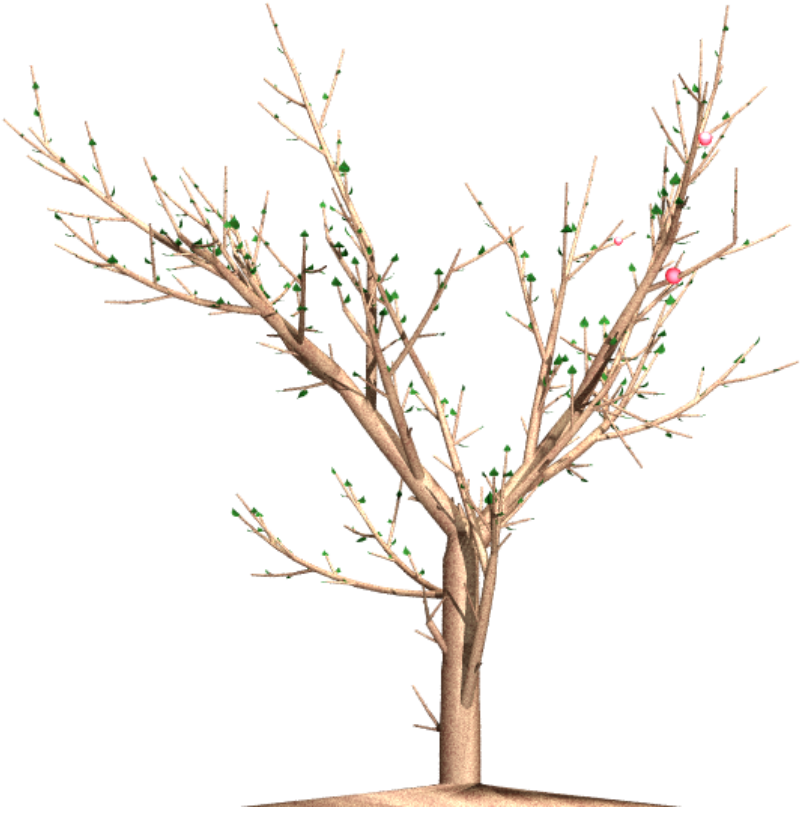
However, if *MOR* is large enough that new radial rings are produced before breaking stress is reached, then reaction wood may be produced in these rings. If deviation from EP exceeds the threshold ξ, the production of reaction wood preempts breakage of the branch by reducing stress as described in Section 8.3. If deviation from EP does not exceed the ξ, reaction wood is not produced and the branch may break at the next step due to the increase in weight of the fruit as it grows.

The breaking strength of branches in trees bearing heavy fruit has a significant impact on the shape of a developed tree. A comparison of the same tree model with high and low breaking strengths is shown in Figure 8.9. In 8.9(a), there is less breakage due to higher breaking strength, and so reaction wood has an opportunity to act in correcting leaning branches. In 8.9(b), breakage occurs before reaction wood can act.

L-system code for the reaction wood and breakage simulations is presented in the Appendix.



(a)



(b)

Figure 8.9: Comparison of the same model with (a) high and (b) low MOR thresholds for breakage.

Chapter 9

Toward a model of tree branch dynamics

The solutions presented so far in this thesis concern the quasi-static problem of determining a tree's equilibrium shape during each stage in its growth. The current chapter investigates a dynamics problem, working toward a model to represent oscillatory motion of tree branches in response to external forces such as the tugging of a branch.

A tree model for dynamic simulation requires a structure that allows the representation of oscillations of a curved shape. To fulfill this requirement and for coherence in this thesis, it is convenient to use the spring-linked segment model described in Chapter 6 for the semi-static case. When using the spring-linked segment model to represent dynamics, the motion of each segment i is described from the perspective of the end of the previous segment $i-1$, instead of being described from a fixed point in space. It is important to note that this reference point can be moving; specifically, it may have a nonzero acceleration.

To describe motion from the perspective of a point that may itself be accelerating, *non-inertial systems* are used. In this chapter, the use of non-inertial systems in several dynamics models is described. Although the final goal is to represent the dynamics of tree branches, the difficult concepts involved in non-inertial systems are introduced by first describing their use in some simpler models. Each system to be presented in this chapter provides a step toward the final goal, as each new system introduces characteristics present in the final model of tree-branch dynamics.

First, a simple mass-spring system is described in which motion occurs along a line. This system is represented in two ways: using first inertial and then non-inertial reference frames. These approaches are compared so as to understand the distinction between them and verify that both representations produce identical results. These systems are presented in Sections

9.2.1 and 9.2.2.

Next, two-dimensional rotational systems are described. Introduced separately into the model are the forces from gravity (in Section 9.3) and rotational springs (in Section 9.4). Finally, the system is extended to branching structures in Section 9.6. Limitations of the system and directions for future work are described in Section 9.7.

9.1 Introduction to non-inertial systems

An object's motion is always described with respect to a specific frame of reference. If this frame of reference is fixed in space or moving at a constant velocity, the reference frame is called *inertial* [6]. Motion viewed from such a reference frame is said to be viewed from an inertial perspective.

However, if the reference frame from which motion in viewed has a changing velocity (a nonzero acceleration), it is referred to as a *non-inertial reference frame*, or accelerating frame [6]. When viewing a moving object from such a frame, the perceived motion is partially due to the acceleration of the reference frame. Therefore when describing an object's motion from such a perspective, one must account for the acceleration of the reference frame.

For example, the motion of a woman sitting in a car that is braking suddenly at a red light can be described non-inertially, from the reference frame of the car itself. From the non-inertial perspective of the car, it appears that the woman, previously at rest, is suddenly moving forward in her seat. However, from an inertial perspective, an observer standing on the street corner would note that the woman's forward motion has actually slowed. Clearly, the accelerations observed from these two perspectives are different. In fact, they are related by the following equation:

$$^{corner}\vec{a}_{woman} = {}^{corner}\vec{a}_{car} + {}^{car}\vec{a}_{woman}, \tag{9.1}$$

where a is acceleration, and the leading superscript describes the reference frame from which

the acceleration is being observed. Addition of the acceleration of the car (in this case, a deceleration) to the acceleration of the woman with respect to the car gives the woman's acceleration as it would be viewed from the street corner. In general, this equation can be rewritten as

$$^{fixed}\vec{a}_{object} = {}^{fixed}\vec{a}_{RF} + {}^{RF}\vec{a}_{object}, \tag{9.2}$$

where *RF* is the reference frame [6].

Equation 9.1 can be manipulated to solve for the woman's acceleration as viewed from the accelerating car:

$$^{car}\vec{a}_{woman} = {}^{corner}\vec{a}_{woman} - {}^{corner}\vec{a}_{car} \tag{9.3}$$

In general terms, the equation for acceleration viewed from a non-inertial reference frame is

$$^{RF}\vec{a}_{object} = {}^{fixed}\vec{a}_{object} - {}^{fixed}\vec{a}_{RF}, \tag{9.4}$$

where $^{RF}\vec{a}_{object}$ is the object's acceleration viewed non-inertially (from the accelerating reference frame), $^{fixed}\vec{a}_{object}$ is the object's acceleration viewed from an inertial perspective and $^{fixed}\vec{a}_{RF}$ is the acceleration of the reference frame, also viewed inertially.

Using Newton's second law of dynamics, Equation 9.4 can also be extended to describe forces:

$$m_{object}\,(^{RF}\vec{a}_{object}) = m_{object}\,(^{fixed}\vec{a}_{object}) - m_{object}\,(^{fixed}\vec{a}_{RF}), \tag{9.5}$$

where m_{object} is the mass of the object. $m_{object}\,(^{fixed}\vec{a}_{RF})$ is a non-inertial force or *fictitious force* [6], which results from the acceleration of the reference frame. Though this force does not exist for the inertial observer, it is needed to describe the force observed from the accelerating reference frame.

Note that when motion is viewed from a non-inertial frame, the object inherits the negation of the reference frame's acceleration, or force. The reason for this negation is evident from the example of the woman in a car. When the moving car stops suddenly (a deceleration

of the reference frame), the woman feels a force pushing her forward. This forward force exists only as seen from the non-inertial reference frame of the moving car: as seen from the inertial perspective of the observer on the street corner, there is no such forward force. The forward force in the non-inertial system comes from the negation of the deceleration of the reference frame. It is an example of a non-inertial force.

9.2 A one-dimensional mass-spring system

This chapter's investigation of non-inertial systems begins with a simple one-dimensional mass-spring simulation. To illustrate the implications of viewing motion from a non-inertial perspective, the same system will be modelled from both inertial and non-inertial reference frames.

The system is made up of a chain of point masses, or particles, linked by springs which can be stretched or compressed along one direction, as depicted in Figure 9.1. The leftmost spring is attached to a fixed point (such as a wall). The combination of a spring with a particle at its distal end is called a *link*. Particle 0 is located at the distal end of link 0, the first link. For convenience, links of lower index will be called *parents* of the higher-index links, or *children*.

9.2.1 The system expressed inertially

First, the inertial system will be described. In this model, the motion of all links is viewed from a common fixed reference frame. For convenience, this reference frame is located on the wall, at the point of attachment of the leftmost spring. This fixed reference frame is labelled W.

The force exerted on particle i by spring i is defined as

$$^{W}F^{on\ i}_{spring_i} = -k_i\ (l_i - \bar{l}_i), \tag{9.6}$$

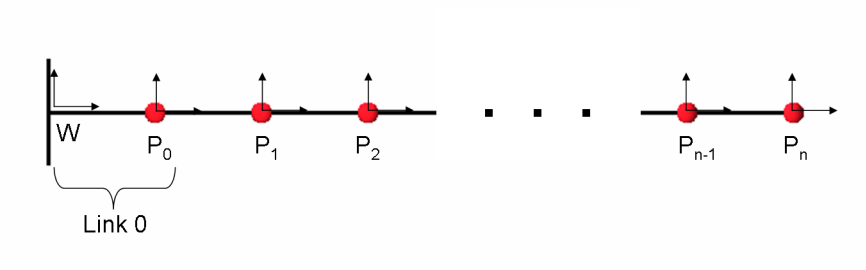


Figure 9.1: Inertial reference frame W and non-inertial reference frames for a one-dimensional mass-spring system.

where l_i is the current length of spring i and $\bar{l}_i$ is its rest length.

A force equal in magnitude but opposite in direction is exerted by spring i on particle $i+1$:

$$^{W}F^{on\ i+1}_{spring_i} = k_i\ (l_i - \bar{l}_i). \tag{9.7}$$

Each particle's acceleration is influenced by its two surrounding springs. Particle i is affected by forces from its two surrounding springs i and $i+1$. On particle i,

$$^{W}F_{spring_i} = -k_i\ (l_i - \bar{l}_i) \tag{9.8}$$

$$^{W}F_{spring_{i+1}} = k_{i+1}\ (l_{i+1} - \bar{l}_{i+1}) \tag{9.9}$$

$$^{W}F_{springs} = {}^{W}F_{spring_i} + {}^{W}F_{spring_{i+1}}. \tag{9.10}$$

A damping force is introduced, which is proportional to the velocity of the particle:

$$^{W}F_{springs} = (^{W}F_{spring_i} + {}^{W}F_{spring_{i+1}}) - b\ (^{W}v_i), \tag{9.11}$$

where b is a damping constant and $^{W}v_i$ is the velocity of particle i as observed from the inertial reference frame W.

Based on this force, particle i's acceleration can be determined from Newton's second law,

$$^{W}a_i = \frac{^{W}F_{springs}}{m_i}, \tag{9.12}$$

where m_i is the mass of particle i. Recall that the notation $^{W}a_i$ describes the acceleration of particle i as observed from the reference frame W, in this case an inertial reference frame.

The inertially-viewed velocity of particle i, ${}^{W}v_i$, can be obtained from the inertially-viewed acceleration ${}^{W}a_i$ by using a numerical technique such as Euler's method:

$$ {}^{W}v_i^{t+1} \;=\; {}^{W}v_i^{t} \;+\; \Delta t\ ({}^{W}a_i^{t}). \tag{9.13} $$

The length of the spring is then determined from the distance between its two surrounding particles $i-1$ and i. Euler's method is used again, using the difference between the two particles' velocities as the derivative:

$$ l_i^{t+1} \;=\; l_i^{t} \;+\; \Delta t\ ({}^{W}v_i^{t} \;-\; {}^{W}v_{i-1}^{t}). \tag{9.14} $$

Using the above-described equations, the calculations for particle i at time $t+1$ in the inertial system are:

$$ \begin{aligned} {}^{W}F_{spring_i}^{t+1} &= -k_i\ (l_i^t - \bar{l}_i^t) \\ {}^{W}F_{spring_{i+1}}^{t+1} &= k_{i+1}\ (l_{i+1}^t - \bar{l}_{i+1}^t) \\ {}^{W}F_{springs}^{t+1} &= ({}^{W}F_{spring_i}^{t+1} \;+\; {}^{W}F_{spring_{i+1}}^{t+1}) \;-\; b\ ({}^{W}v_i^t) \\ {}^{W}a_i^{t+1} &= \frac{{}^{W}F_{springs}^{t+1}}{m_i} \\ {}^{W}v_i^{t+1} &= {}^{W}v_i^{t} \;+\; \Delta t\ ({}^{W}a_i^{t}) \\ l_i^{t+1} &= l_i^{t} \;+\; \Delta t\ ({}^{W}v_i^{t} \;-\; {}^{W}v_{i-1}^{t}). \end{aligned} $$

9.2.2 The system expressed non-inertially

An equivalent model to the one above may be created using non-inertial reference frames. Instead of using a single fixed reference frame, the motion of each particle i may be observed from the reference frame located at the previous particle $i-1$. These reference frames are depicted in Figure 9.1. Because each particle may be accelerating, this system is non-inertial.

According to Equation 9.4, the acceleration of particle i viewed from the non-inertial reference frame at particle $i-1$ is:

$$ {}^{i-1}a_i \;=\; {}^{W}a_i \;-\; {}^{W}a_{i-1}, \tag{9.15} $$

where the superscript ${}^{i-1}$ indicates that the reference frame is located at particle $i-1$. Equivalently,

$$ {}^{W}a_i \;=\; {}^{W}a_{i-1} \;+\; {}^{i-1}a_i. \tag{9.16} $$

However, the acceleration ${}^{W}a_{i-1}$ as viewed from the inertial frame may be unknown. If $i > 1$, the reference frame at $i-1$ is viewed from another particle $i-2$, which may itself be accelerating.

Links are attached in series, so from an inertial perspective each particle inherits the acceleration of its parent. Only link 0 is attached to a wall, so it does not inherit any acceleration. Particle 0's acceleration is already described inertially, as it is viewed from the stationary reference frame W. Particle 1 has its own acceleration relative to particle 0, which when added to particle 0's acceleration gives the inertially-viewed acceleration for link 1, as viewed from W. To get particle 2's inertially-viewed acceleration, its acceleration relative to particle 1 is added to the inertially-viewed acceleration of particle 1. Thus, non-inertial accelerations from the links of lower index (links 0 .. $i-1$) are accumulated to obtain the inertially-viewed acceleration of particle i. Summation of these accelerations gives ${}^{W}a_{i-1}$, so Equation 9.15 becomes

$$ {}^{i-1}a_i \;=\; {}^{W}a_i \;-\; \sum_{j=0}^{i-1} {}^{j-1}a_j. \tag{9.17} $$

The use of non-inertial systems changes several aspects of the equations given above. According to Equation 9.5, to specify spring forces from a non-inertial reference frame requires a fictitious force $m_i\,({}^{W}a_{i-1})$ from the acceleration of the reference frame at $i-1$:

$$ {}^{i-1}F_{springs} \;=\; ({}^{W}F_{spring_i} \;+\; {}^{W}F_{spring_{i+1}}) \;-\; b\,({}^{W}v_i) \;-\; m_i\,({}^{W}a_{i-1}). \tag{9.18} $$

Also, Equation 9.14 for updating spring length no longer needs to access two velocities, because

$$ {}^{i-1}v_i \;=\; ({}^{W}v_i \;-\; {}^{W}v_{i-1}), \tag{9.19} $$

as in Equation 9.4. So Equation 9.14 becomes simply

$$l_i^{t+1} \;=\; l_i^t \;+\; \Delta t\ ({}^{i-1}v_i^t). \tag{9.20}$$

Finally, inertially-specified acceleration and velocities must be calculated according to Equation 9.17, for use in the fictitious force and damping terms of Equation 9.18. The final calculations for particle i at time $t+1$ in the non-inertial system are:

$$\begin{aligned}
{}^{W}F_{spring_i}^{t+1} &= -k_i\ (l_i^t - \bar{l}_i^t) \\
{}^{W}F_{spring_{i+1}}^{t+1} &= k_{i+1}\ (l_{i+1}^t - \bar{l}_{i+1}^t) \\
{}^{i-1}F_{springs}^{t+1} &= ({}^{W}F_{spring_i}^{t+1} \;+\; {}^{W}F_{spring_{i+1}}^{t+1}) \;-\; b\ ({}^{W}v_i^t) \;-\; m_i\ ({}^{W}a_{i-1}^t) \\
{}^{i-1}a_i^{t+1} &= \frac{{}^{i-1}F_{springs}^{t+1}}{m_i} \\
{}^{i-1}v_i^{t+1} &= {}^{i-1}v_i^t \;+\; \Delta t\ ({}^{i-1}a_i^t) \\
l_i^{t+1} &= l_i^t \;+\; \Delta t\ ({}^{i-1}v_i^t) \\
{}^{W}a_i^{t+1} &= {}^{W}a_{i-1}^{t+1} \;+\; {}^{i-1}a_i^{t+1} \\
{}^{W}v_i^{t+1} &= {}^{W}v_{i-1}^{t+1} \;+\; {}^{i-1}v_i^{t+1}.
\end{aligned}$$

As desired, tests of inertial and non-inertial mass-spring systems indicate that they produce identical resulting motion.

9.3 A non-inertial system for rotation – multiply-linked pendula

The mass-spring system described in Section 9.2 models springs that stretch in one dimension. However, this stretching of a one-dimensional axis does not resemble the motion of swaying a tree branch. To progress toward this goal, the model must better simulate the structure and motion of a tree branch, which can arch and oscillate rotationally, but does not stretch in its length. To achieve this aim, a rotational non-inertial system has been developed. This system is comprised of a chain of straight weightless segments which cannot stretch or

compress in length. As in the one-dimensional model, a point mass is located at the distal end of each segment. Rotations can occur at the joints between segments, and these joint rotations account for the arching and oscillation of the chain.

A system consisting of such a chain of segments lying in the plane will be described. In this system, all rotations occur around axes perpendicular to the plane. The first model, described in Section 9.3.2, represents the motion of the system with only gravity acting on the particles. This system is therefore one of linked pendula. In Section 9.4, rotational springs located at the joints are added to the existing system. These springs act to maintain a relative rest angle between two consecutive segments. Finally, the model is extended to represent branching structures.

As in Section 9.2, non-inertial reference frames are used to describe motion in the system. Particle 0's motion is viewed from the inertial reference frame of the wall, but every other particle i's motion is observed from the reference frame located at the preceding particle $i-1$. The system is no longer one-dimensional, so the orientation of these reference frames is important. For simplicity, their orientations are defined to be identical to that of the inertial frame: that is, the reference frames do not rotate along with the links. The orientation of each link with respect to the reference frame is therefore defined by the angle α that its axis forms with respect to the horizontal, as shown in Figure 9.2. Each link axis r_i is a vector directed from the centre of its reference frame (at particle $i-1$) to its own particle, i.

The equations required to represent the system of Figure 9.2 are described in Section 9.3.2. First, some concepts needed to derive these equations are introduced.

9.3.1 Radial and tangential components of forces

A force can be specified as a vector: it is defined by a direction and a magnitude. In the discussion that follows, it is important to note that any vector $\vec{v}$ can be decomposed into its perpendicular and parallel components with respect to any other vector $\vec{w}$. In particular, a

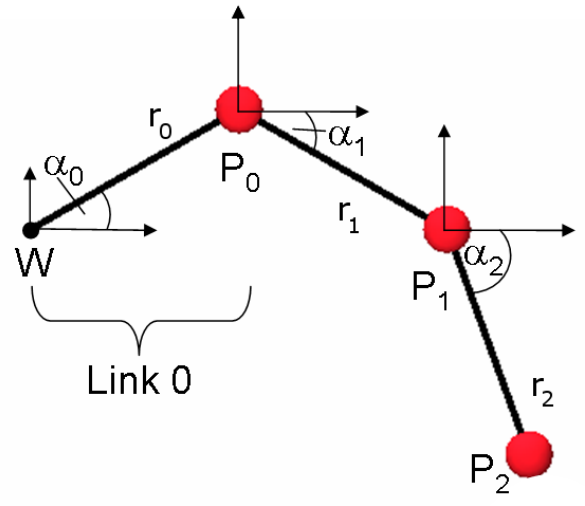


Figure 9.2: Rotational system of three linked pendula.

force vector $\vec{F}$ can be resolved into its parallel and perpendicular components $\vec{F}_{\|}$ and $\vec{F}_{\perp}$ with respect to the axis $\vec{r}_i$ of the link upon which $\vec{F}$ acts.

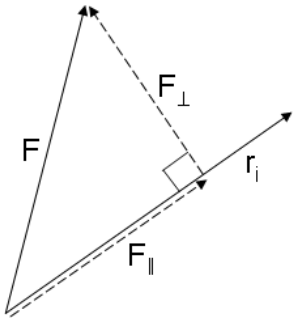


Figure 9.3: Decomposition of a force vector $\vec{F}$, with respect to a reference vector $\vec{r}_i$.

Using vector notation, obtaining the parallel and perpendicular components of $\vec{F}$ with respect to $\vec{r}_i$ is equivalent to projecting $\vec{F}$ onto the unit reference vector $\hat{r}_i = \frac{\vec{r}_i}{\|\vec{r}_i\|}$ and onto a perpendicular unit vector, $\hat{r_{i_\perp}}$. These operations are defined as:

$$\vec{F}_{\|} = (\hat{r}_i \cdot \vec{F}) * \hat{r}_i \tag{9.21}$$

$$\vec{F}_{\perp} = -((\vec{F} \times \hat{r}_i) \times \hat{r}_i). \tag{9.22}$$

Decomposition of any force $\vec{F}$ acting on link i into its perpendicular (*tangential*) and parallel (*radial*) components with respect to the axis $\vec{r}_i$ of link i effectively separates forces

that cause rotation from forces that act along a link to stretch or compress it. As explained below, purely tangential force can only cause rotation, and purely radial force can only cause stretching.

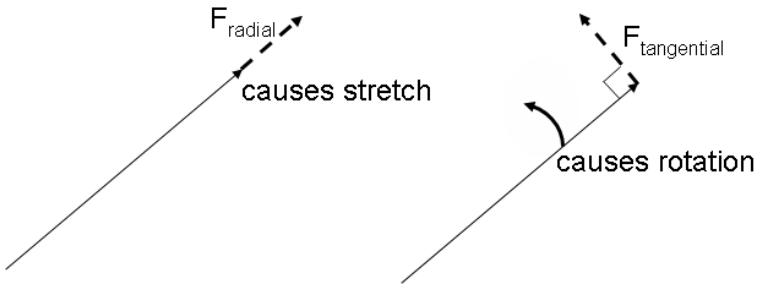


Figure 9.4: Radial force component causes stretching and tangential component causes rotation.

Because segments in this system are of fixed length, they cannot be stretched or compressed no matter what force acts upon them. Therefore any stretching force $\vec{F}$ acting on link i has no effect on the link itself. The pushing or pulling that such a force $\vec{F}$ causes may result in linear motion of the whole link, but only as a result of $\vec{F}$ causing the rotation of another link earlier in the chain. Therefore, when computing the result of a force $\vec{F}$ acting on link i, we can disregard the radial component $\vec{F}_{||}$ with respect to $\vec{r}_i$ and consider only the tangential (rotation-causing) component.

However, this unused radial component $\vec{F}_{||}$ must not be discarded altogether. It is an unused force in the system which could cause rotation of another differently-oriented segment in the chain. It is important to remember that components of $\vec{F}$ are taken with respect to a particular link axis $\vec{r}_i$. A force that is radial with respect to link i is not necessarily radial with respect to link $i-1$. This force will have no rotational effect on i, but it will rotate link $i-1$, as long as it is not parallel to i.

Figure 9.5 depicts force $\vec{F}_2$, a force which acts on links of index $i < 2$ as a result of particle 2's outward acceleration. Because link 1 is parallel to link 2, $\vec{F}_2$ is entirely radial with respect to link 1, and therefore has no effect on it. If link 0 were also parallel to the

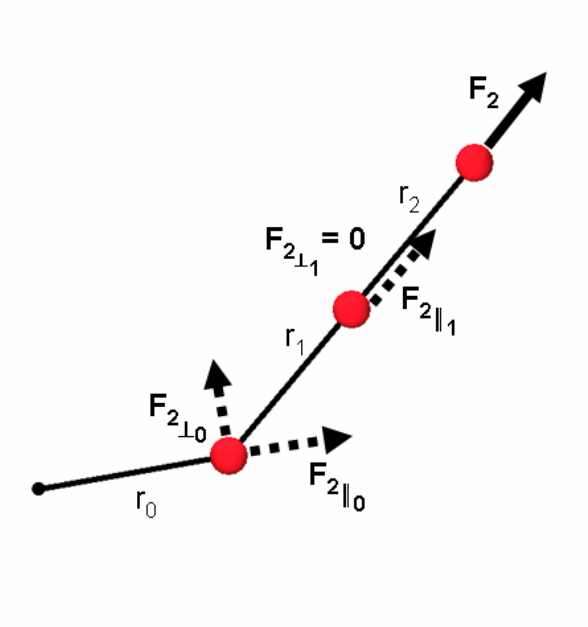


Figure 9.5: Three links, the second two parallel, with components of force $\vec{F}_2$ shown with respect to each link $\vec{r}_i$. Only $\vec{F_{2_{\perp_0}}}$ causes rotation, as it is the only nonzero tangential component of $\vec{F}_2$.

other two links, $\vec{F}_2$ would have no effect on the system, as the segments cannot stretch and the system is attached to a fixed point at link 0's proximal end. However, in the scenario depicted by this figure, $\vec{F}_2$ causes a rotation at link 0, because it has a nonzero tangential component $\vec{F_{2_{\perp_0}}}$ with respect to the axis $\vec{r}_0$ of link 0. The result will be the pulling of the entire chain in the direction of $\vec{F}_2$, because child links 1 and 2 are attached to link 0 and so will necessarily follow.

To model the effect of radial forces on segments of lower index, the unused radial components of forces must not be discarded, but must be passed on to parent segments which they might affect. At each link i, forces that can affect particle i are accumulated, and the resulting total force is decomposed into tangential and radial components. The tangential component is used immediately to cause rotation, and the unused radial component is passed on to link $i-1$.

It is important to note that when the unused radial force is passed on to links of lower

index, it must be stored as a vector in the inertial frame. A non-inertially viewed vector cannot be passed on for use in another non-inertial reference frame, because it assumes a relationship to the reference frame that no longer holds. However, an inertially-viewed vector can be used from any frame.

9.3.2 Overview of the equations for the rotational system

In this section, the equations for the rotational system are described. First, an overview of the system's equations is presented, and later the equations will be examined in detail.

First, all the forces acting on the particle at the end of link i are accumulated to obtain the total force $\vec{F}_i^{t+1}$ acting on particle i at time $t+1$.

$$\vec{F}_i^{t+1} = \sum_j \vec{f}_{component_j}^{t+1} \tag{9.23}$$

Each of these force contributions $\vec{f}_{component_j}^{t+1}$ will be described in Section 9.3.3.

This total force $\vec{F}_i^{t+1}$ acts on the rotational system in two ways. First, its tangential component $\vec{F}_{i_\perp}^{t+1}$ determines the rotational acceleration of link i, $\vec{\varepsilon}_i^{t+1}$. The value of $\vec{\varepsilon}_i^{t+1}$ that accounts for the acceleration of previous links depends on ${}^{i-1}\vec{F}_{i_\perp}^{t+1}$, the tangential force as viewed from the non-inertial reference frame located at link $i-1$, according to

$${}^{i-1}\vec{\varepsilon}_i^{t+1} = \zeta({}^{i-1}\vec{F}_{i_\perp}^{t+1}), \tag{9.24}$$

where ζ is a function to be detailed below.

Second, the radial component of $\vec{F}_i^{t+1}$ will be used to cause motion in the other links of lower index. The portion of $\vec{F}_{i_\parallel}^{t+1}$ which affects these other links is computed by subtracting the force that caused the *radial* acceleration of link i itself. Specifically, the force to be inherited by link $i-1$ is

$${}^{W}\vec{F}_{toLeft_i}^{t+1} = {}^{W}\vec{F}_{i_\parallel}^{t+1} - (m_i \ {}^{W}\vec{a}_{i_\parallel}^{t+1}) \tag{9.25}$$

where $(m\ {}^{W}\vec{a}_{i_\parallel}^{t+1})$ is the force used to cause the radial acceleration ${}^{W}\vec{a}_{i_\parallel}^{t+1}$ of link i. This acceleration is viewed inertially, so $\vec{F}_{i_\parallel}^{t+1}$ must be viewed from an inertial perspective, expressed as ${}^{W}\vec{F}_{i_\parallel}^{t+1}$.

9.3.3 The rotational system's equations in detail

The force contributions to ${}^{W}\vec{F}_{i}^{t+1}$ are now examined in detail. These include the force from gravity, forces left over from links further to the right, and damping force.

$$ {}^{W}\vec{F}_{i}^{t+1} \;=\; {}^{W}\vec{f}_{grav_i}^{t+1} \;+\; {}^{W}\vec{f}_{fromRight_i}^{t+1} \;+\; {}^{W}\vec{f}_{damp_i}^{t+1}. \tag{9.26} $$

Note that, according to Equation 9.5, all of these contributions must be stated as viewed from an inertial frame W.

Gravity acts upon link i's mass according to

$$ {}^{W}\vec{f}_{grav_i}^{t+1} = m_i\,\vec{g}, \tag{9.27} $$

where $\vec{g} = (0, -9.8, 0)\ ms^{-2}$ and m_i is the mass at the distal end of link i. As desired, ${}^{W}\vec{f}_{grav_i}^{t+1}$ is the force that would be observed from an inertial reference frame, because the vector $\vec{g}$ is described inertially.

Next, the effect on link i of unused radial forces from child links $i+1$ to n is

$$ {}^{W}\vec{f}_{fromRight_i}^{t+1} = \sum_{k=i+1}^{n} {}^{W}\vec{f}_{k}^{t+1}, \tag{9.28} $$

where each ${}^{W}\vec{f}_{k}^{t+1}$ represents the radial force acting on the mass at link k, minus the force required to cause the *radial* acceleration of all links between k and i,

$$ {}^{W}\vec{f}_{k}^{t+1} \;=\; {}^{W}\vec{F}_{k_\parallel}^{t+1} - \sum_{j=i+1}^{k} (m_j\ {}^{W}\vec{a}_{j_\parallel}^{t+1}). \tag{9.29} $$

The damping force, ${}^{W}\vec{F}_{damp_i}^{t+1}$, inhibits rotational acceleration of link i, and is approximated as

$$ {}^{W}\vec{F}_{damp_i}^{t+1} = b\ {}^{W}\vec{\omega}_{i}^{\,t} \;\times\; \vec{r}_{i}^{\,t}, \tag{9.30} $$

where b is a user-controlled damping constant, ${}^{W}\vec{\omega}_i^{\,t}$ is the inertially specified rotational velocity of link i during the previous time step t and $\vec{r}_i^{\,t}$ is the vector representing the long axis of link i, also at time t. $b\,{}^{W}\vec{\omega}_i^{\,t}$ is the damping torque, and the cross-product results in a damping force.

Making use of all these values, the resulting force ${}^{W}\vec{F}_i^{t+1}$ of link i is

$$ {}^{W}\vec{F}_i^{t+1} \;=\; {}^{W}m_i\,\vec{g} \;+\; \sum_{k=i+1}^{n} {}^{W}\vec{f}_k^{t+1} \;-\; m_i\,b\,{}^{W}\vec{\omega}_i^{\,t}\times\vec{r}_i^{\,t}. \tag{9.31} $$

The total force viewed from a non-inertial reference frame

If $\vec{F}_i^{t+1}$ is viewed from a non-inertial reference frame, the acceleration of the reference frame itself must be accounted for. This is done by subtracting a force $\vec{f}_{RF_i}^{t+1}$ obtained from the acceleration of the reference frame, as described in Equation 9.5,

$$ {}^{i-1}\vec{F}_i^{t+1} \;=\; {}^{W}\vec{F}_i^{t+1} \;-\; {}^{W}\vec{f}_{RF_i}^{t+1}. \tag{9.32} $$

This force $\vec{f}_{RF_i}^{t+1}$ is

$$ {}^{W}\vec{f}_{RF_i}^{t+1} = m_i\,{}^{W}\vec{a}_{i-1}^{t+1}, \tag{9.33} $$

where ${}^{W}\vec{a}_{i-1}^{t+1}$ is the inertially-viewed acceleration of the reference frame, located at the end of link $i-1$.

${}^{i-1}\vec{F}_i^{t+1}$ from Equation 9.5 can be resolved into its radial and tangential components ${}^{i-1}\vec{F}_{i_\parallel}^{t+1}$ and ${}^{i-1}\vec{F}_{i_\perp}^{t+1}$ with respect to the link direction $\vec{r}_i^{\,t+1}$. These forces affect the system in two ways, as outlined in Section 9.3.2.

First, the tangential force ${}^{i-1}\vec{F}_{i_\perp}^{t+1}$ directly produces a rotational acceleration for link i as follows:

$$ {}^{i-1}\vec{\varepsilon}_i^{\,t+1} = \frac{(\vec{r}_i^{\,t} \;\times\; {}^{i-1}\vec{F}_{i_\perp}^{t+1})}{m|\vec{r}_i^{\,t}|^2}. \tag{9.34} $$

This equation was derived as follows. An object's rotational acceleration $\vec{\varepsilon}$ resulting from a torque $\vec{\tau}$ can be obtained by dividing by the object's moment of inertia I. For a weightless bar with a mass at its distal end (a pendulum), $I = m|\vec{r}|^2$ [52]. In the following derivation for

a pendulum, subscripts and superscripts are removed for clarity:

$$\varepsilon = \frac{\vec{\tau}}{I} \tag{9.35}$$

$$= \frac{\vec{\tau}}{m|\vec{r}|^2} \tag{9.36}$$

$$= \frac{(\vec{r}\times\vec{F})}{m|\vec{r}|^2}. \tag{9.37}$$

Because by definition the cross-product $\vec{r}\times\vec{F}$ uses only the component of $\vec{F}$ perpendicular to $\vec{r}$, the total force $\vec{F}_i$ can be used in Equation 9.37, instead of taking the perpendicular component first:

$${}^{i-1}\vec{\varepsilon}_i{}^{t+1} = \frac{(\vec{r}_i^{\,t} \times {}^{i-1}\vec{F}_i^{t+1})}{m|\vec{r}_1^{\,t}|^2}. \tag{9.38}$$

The above simplification produces the same result as Equation 9.34, and saves computation time. The rotational acceleration ${}^{i-1}\vec{\varepsilon}_i{}^{t+1}$ can be integrated to obtain the rotational velocity ${}^{i-1}\vec{\omega}_i{}^{t+1}$ and the current angle $({\alpha_i}^{t+1}).z$ of the segment relative to the horizontal. The numerical method used to perform this integration will be discussed in Section 9.3.5. The link axis $\vec{r}_i^{\,t}$ is then rotated by the angle $({\alpha_i}^{t+1}).z$ to obtain the updated link axis at time $t+1$, $\vec{r}_1{}^{t+1}$.

Secondly, this force ${}^{i-1}\vec{F}_i^{t+1}$ affects links to the left. As the perpendicular component ${}^{i-1}\vec{F}_{i_\perp}^{t+1}$ has been used up entirely in causing rotation, only the parallel component ${}^{i-1}\vec{F}_{i_\parallel}^{t+1}$ is left to affect these links of lower index. However, before passing ${}^{i-1}\vec{F}_{i_\parallel}^{t+1}$ to the left, the effect of this force on link *i* itself must be subtracted.

From a non-inertial perspective, ${}^{i-1}\vec{F}_{i_\parallel}^{t+1}$ does not appear to have any effect on link *i*: it cannot cause *i*'s rotation, nor can it stretch or compress the link *i* itself. Thus, if the motion is viewed from the non-inertial reference frame at the end of link $i-1$, ${}^{i-1}\vec{F}_{i_\parallel}^{t+1}$ does not cause any acceleration at link *i*. However, from an inertial perspective link *i* may indeed be accelerating as a result of ${}^{W}\vec{F}_{i_\parallel}^{t+1}$. This force can cause rotation of a parent link, which will result in an acceleration of link *i* when viewed inertially, as in Figure 9.5. To subtract the effect of this acceleration,the radial calculations must be viewed from an inertial frame.

Before passing this force ${}^{W}\vec{F}^{t+1}_{i_{\parallel}}$ to the parent links, the portion of this force which caused the acceleration of link i must be subtracted:

$$ {}^{W}\vec{f}^{t+1}_{toLeft} = {}^{W}\vec{F}^{t+1}_{i_{\parallel}} - (m_i \, {}^{W}\vec{a}^{t+1}_{i_{\parallel}}). \tag{9.39} $$

${}^{W}\vec{f}^{t+1}_{toLeft}$ is entirely radial with respect to the current link, but when acting on link $i-1$ it will be divided into components based on link $i-1$'s direction. As before, the tangential component is used in rotation. If the system has more than two pendula, the radial component is passed on to link $i-2$ after subtracting the force used to cause link $i-1$'s inertially-viewed radial acceleration.

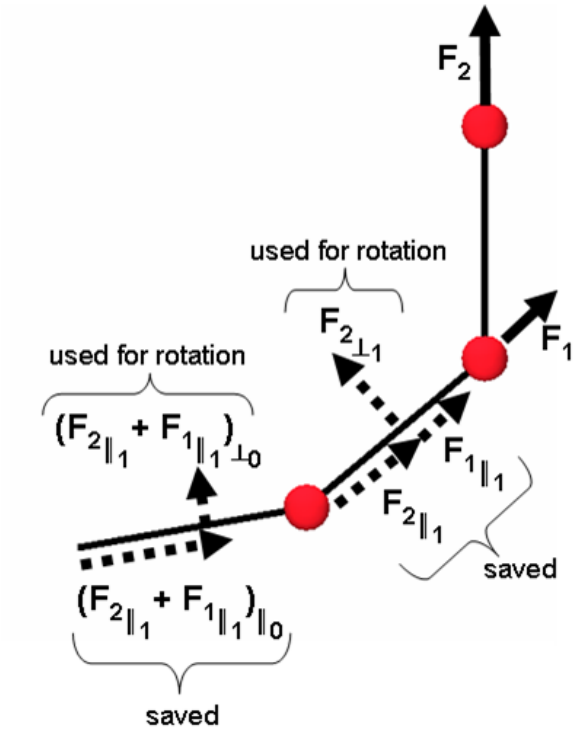


Figure 9.6: Accumulated radial forces from the right, taking perpendicular and parallel components at each step.

All that remains is to determine the final linear acceleration of link i. From the non-inertial perspective, both tangential and radial components of linear acceleration can be ob-

tained directly from the rotational values:

$$ {}^{i-1}\vec{a}_{i_{\parallel}}^{\,t+1} = {}^{i-1}\vec{\omega}_i^{\,t+1} \times ({}^{i-1}\vec{\omega}_i^{\,t+1} \times \vec{r}_i^{\,t+1}) \tag{9.40} $$

$$ {}^{i-1}\vec{a}_{i_{\perp}}^{\,t+1} = {}^{i-1}\vec{\varepsilon}_i^{\,t+1} \times \vec{r}_i^{\,t+1}, \tag{9.41} $$

where ${}^{i-1}\vec{a}_{i_{\perp}}^{\,t+1}$ is the linear acceleration in a direction tangent to the link, and ${}^{i-1}\vec{a}_{i_{\parallel}}^{\,t+1}$ is the *centripetal* acceleration of the distal point of link *i*. Centripetal acceleration is the inward linear acceleration required to constrain the link's acceleration to a circle – it results simply from the rod being of fixed length while tangential acceleration is going on. Both accelerations are depicted in Figure 9.7.

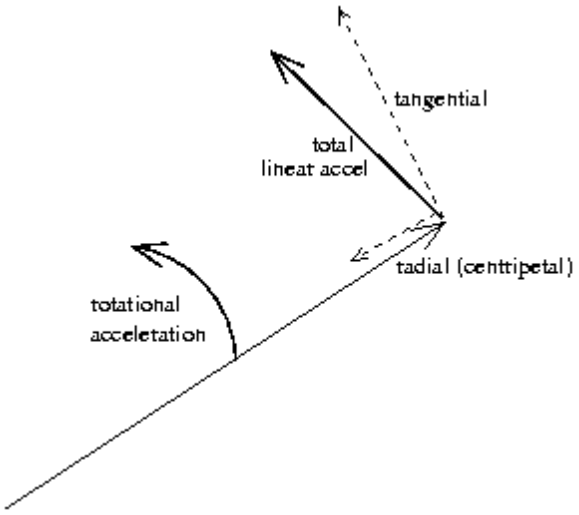


Figure 9.7: Centripetal and tangential accelerations.

To obtain the acceleration of link *i* as observed from an inertial perspective, the two components ${}^{i-1}\vec{a}_{i_{\parallel}}^{\,t+1}$ and ${}^{i-1}\vec{a}_{i_{\perp}}^{\,t+1}$ are summed and added to the inertially-viewed acceleration of the reference frame, ${}^{W}\vec{a}_{i-1}^{\,t+1}$:

$$ {}^{W}\vec{a}_i^{\,t+1} = {}^{W}\vec{a}_{i-1}^{\,t+1} + {}^{i-1}\vec{\omega}_i^{\,t+1} \times ({}^{i-1}\vec{\omega}_i^{\,t+1} \times \vec{r}_i^{\,t+1}) + {}^{i-1}\varepsilon_i^{\,t+1} \times \vec{r}_i^{\,t+1}. \tag{9.42} $$

9.3.4 Summary of the equations

In summary, the equations that have been developed for this system of linked pendula are:

$$ {}^{W}\vec{f}_{k}^{\,t+1} = {}^{W}\vec{F}_{k_{\parallel}}^{\,t+1} - \sum_{j=i+1}^{k} (m_j \, {}^{W}a_{j_{\parallel}}^{t+1}) \tag{9.43} $$

$$ {}^{W}\vec{F}_{i}^{\,t+1} = {}^{W}m_i \, \vec{g} \;+\; \sum_{k=i+1}^{n} {}^{W}\vec{f}_{k}^{\,t+1} \;-\; m_i \, b \, {}^{W}\vec{\omega}_i^{\,t} \times \vec{r}_i^{\,t} \tag{9.44} $$

$$ {}^{i-1}\vec{F}_{i}^{\,t+1} = {}^{W}\vec{F}_{i}^{\,t+1} - {}^{W}\vec{f}_{RF_i}^{\,t+1} \tag{9.45} $$

$$ {}^{i-1}\varepsilon_{i}^{t+1} = \frac{(\vec{r}_i^{\,t} \times {}^{i-1}\vec{F}_{i}^{\,t+1})}{m|\vec{r}_i^{\,t}|^2} \tag{9.46} $$

$$ {}^{W}\vec{a}_{i}^{\,t+1} = {}^{W}\vec{a}_{i-1}^{\,t+1} + {}^{i-1}\vec{\omega}_i^{\,t+1} \times ({}^{i-1}\vec{\omega}_i^{\,t+1} \times \vec{r}_i^{\,t+1}) + {}^{i-1}\varepsilon_{i}^{t+1} \times \vec{r}_i^{\,t+1} \tag{9.47} $$

$$ {}^{W}\vec{f}_{toLeft}^{\,t+1} = {}^{W}\vec{F}_{i_{\parallel}}^{\,t+1} - (m_i \, {}^{W}\vec{a}_{i_{\parallel}}^{\,t+1}). \tag{9.48} $$

9.3.5 Implementation

In this section, the use of Equations 9.43 to 9.48 in the L-system model is examined.

As described in Section 9.3.2, the unused component of the force ${}^{W}\vec{F}_{i_{\parallel}}^{\,t+1}$ from link i acts on link $i-1$ and further lower-numbered links. However, is does not act on higher-numbered links, because of the way dependency is set up in the model: link $i-1$ is influenced by link i only by inheriting i's acceleration. In contrast, any effect that link i feels from link $i+1$ arises from its unused radial force. Thus, accelerations must be passed toward links of higher index, and unused radial forces must be passed toward links of lower index.

A two-way flow of information can be accomplished using an L-system's forward and backward passes [28]. The equations developed in Section 9.3.2 can be divided into those that require information from links of lower index (forward pass) and those that depend on information from links of higher index (backward pass).

During the model's forward pass, calculation proceeds from the leftmost link 0 to the rightmost link n. Link i inherits the acceleration of link $i-1$ to obtain its own inertially-viewed acceleration ${}^{W}\vec{a}_{i}^{\,t+1}$. The inherited acceleration is also used to calculate the force acting at i, viewed non-inertially. This force ${}^{i-1}\vec{F}_{i}^{\,t+1}$ is then used to obtain the rotational

acceleration ${}^{i-1}\varepsilon_i^{t+1}$. The simplest integration technique, Euler's method, is used to obtain the values ${}^{i-1}\vec{\omega}_i{}^{t+1}$ and ${}^{i-1}\alpha_i^{t+1}$ from ${}^{i-1}\varepsilon_i^{t+1}$. Consequences of the use of Euler's method are discussed in Section 9.7.

During the backward pass, calculation proceeds from the rightmost link n to the leftmost link 0. Link i obtains the unused radial portion of link $i+1$'s acceleration as a force ${}^{W}\vec{f}_k^{t+1}$,

$$ {}^{W}\vec{f}_k^{t+1} \;=\; {}^{W}\vec{F}_{k_{\|}}^{t+1} - \sum_{j=i+1}^{k} (m_j \, {}^{W}a_{j_{\|}}^{t+1}). \tag{9.49} $$

Such unused radial components have been accumulated from links further to the right, and at each link the tangential component of this accumulated force has been removed and used to cause rotation. Thus, each link has used the portion that is tangential to its own axis of all unused forces from the right, and has passed the unused radial portion to its parent as a force. At link i, the accumulated unused forces from all the links to the right is therefore

$$ {}^{W}\vec{f}_{fromRight_i}^{t+1} = \sum_{k=i+1}^{n} {}^{W}\vec{f}_k^{t+1}. \tag{9.50} $$

This force is added to the other force contributions acting on the current link.

The calculations required for the forward and backward passes of the simulation are given below.

Forward pass: for $i = 0$ to n

$$ {}^{i-1}\vec{F}_i^{t+1} \;=\; {}^{W}\vec{F}_i^{t+1} \;-\; (\; m_i \, {}^{W}\vec{a}_{i-1}^{t+1} \;) \tag{9.51} $$

$$ {}^{i-1}\varepsilon_i^{t+1} \;=\; \frac{(\vec{r}_i^{\,t} \;\times\; {}^{i-1}\vec{F}_i^{t+1})}{m|\vec{r}_i^{\,t}|^2} \tag{9.52} $$

$$ {}^{i-1}\vec{\omega}_i{}^{t+1} \;=\; {}^{i-1}\vec{\omega}_i{}^{t} \;+\; {}^{i-1}\varepsilon_i^{t}\Delta t \tag{9.53} $$

$$ \alpha_i^{t+1} \;=\; \alpha_i^{t} \;+\; {}^{i-1}\vec{\omega}_i{}^{t}\Delta t \tag{9.54} $$

$$ \vec{r}_i{}^{t+1} \;=\; \text{Rotate}(\; \vec{r}_i^{\,t},\; (\alpha_i^{t+1}).z \;) \tag{9.55} $$

$$ {}^{W}\vec{a}_i^{t+1} \;=\; {}^{W}\vec{a}_{i-1}^{t+1} \;+\; {}^{i-1}\vec{\omega}_i{}^{t+1} \times ({}^{i-1}\vec{\omega}_i{}^{t+1} \times \vec{r}_i{}^{t+1}) \;+\; (\; {}^{i-1}\varepsilon_i^{t+1} \times \vec{r}_i{}^{t+1} \;) \tag{9.56} $$

Backward pass: for $i = n$ down to 0

$$
\begin{aligned}
{}^{W}\vec{f}^{\,t+1}_{fromRight_i} &= {}^{W}\vec{f}^{\,t+1}_{fromLeft_{i+1}} && (9.57)\\
{}^{W}\vec{F}_i^{\,t+1} &= m_i\vec{g} + {}^{W}\vec{f}^{\,t+1}_{fromRight_i} - (b\,{}^{W}\vec{\omega}_i^{\,t+1} \times \vec{r}_i^{\,t+1}) && (9.58)\\
{}^{W}\vec{f}^{\,t+1}_{toLeft_i} &= ({}^{W}\vec{F}_i^{\,t+1} - m_i\,{}^{W}\vec{a}_i^{\,t+1})_{\parallel} && (9.59)
\end{aligned}
$$

A single time step consists of one forward pass followed by one backward pass.

9.4 Addition of springs to the system

At equilibrium, the system of linked pendula described above will lie in a vertical chain of links hanging down due to gravity below a fixed point. To better simulate a tree branch that has a specific shape at equilibrium and oscillates when forces cause it to deviate from this shape, rotational springs are introduced into the model. These springs introduce another influence on each link – the torque created by the spring's effort to maintain the rest angle of a joint between two links.

Each joint has a rest angle $\overline{\theta}$ describing the equilibrium shape of its part of the branch. The spring's action counteracts deviations from $\overline{\theta}$. If $\overline{\theta}$ is zero, the joint attempts to keep the two links in a straight line. If it is nonzero, the spring will attempt to maintain this relative angle. Once springs have been added to the system, its equilibrium state will be a balance between the forces of gravity and the springs as in Figure 9.8.

The spring labelled $spring_i$ is located at the proximal end of link i; in other words, at the joint between link $i-1$ and link i. $spring_i$ creates a torque acting on each of its surrounding links $i-1$ and i:

$$
\begin{aligned}
\vec{\tau}_{spring_{toRight}} &= -k(\theta - \overline{\theta}) && (9.60)\\
\vec{\tau}_{spring_{toLeft}} &= k(\theta - \overline{\theta}), && (9.61)\\
& && (9.62)
\end{aligned}
$$

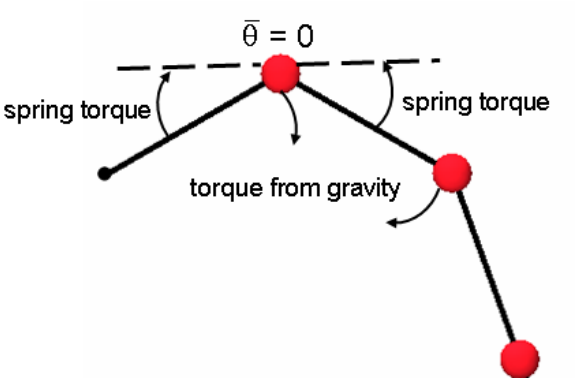


Figure 9.8: Effects of a rotational spring located at particle 0 with $\overline{\theta} = 0$, as well as torques due to gravity.

where k is the spring's stiffness constant and $\theta = (\alpha_i - \alpha_{i-1})$ is the relative angle between the two consecutive segments. This is the rotational version of Hooke's law and is used to compute the torque of the spring due to deviation from rest angle $\overline{\theta}$.

Thus, a link i in the centre of the chain will experience two torques due to its two surrounding springs $spring_i$ and $spring_{i+1}$:

$$\begin{aligned} \vec{\tau}_{leftSpring_i} &= -k_{spring_i}(\theta_i - \overline{\theta_i}) \\ \vec{\tau}_{rightSpring_i} &= k_{spring_{i+1}}(\theta_{i+1} - \overline{\theta_{i+1}}) \\ \vec{\tau}_{springs_i} &= \vec{\tau}_{leftSpring_i} + \vec{\tau}_{rightSpring_i}. \end{aligned}$$

To add springs to the system, the spring torque $\vec{\tau}_{springs_i}$ must be incorporated into the equation for total acceleration ${}^{W}a_i^{t+1}$ of the link; thus it must be expressed as an acceleration instead of a torque. To obtain the acceleration that results from $\vec{\tau}_{springs_i}$, a derivation similar to that of Equations 9.35 - 9.37 is used:

$$\vec{a}_{springs_i}^{t+1} = \frac{\vec{\tau}_{springs_i}^{t}}{I} \times \vec{r}_i^{\,t}. \tag{9.63}$$

In Equation 9.63, rotational acceleration is derived from torque by dividing by the moment of inertia I, where $I = m|\vec{r}|^2$, as described in Section 9.3.3. The corresponding tangential linear acceleration is then obtained by taking the cross-product with the link vector $\vec{r}_i$.

With the addition of the acceleration due to the spring, the equation for the total force from link i becomes

$$ {}^{W}\vec{F}_i^{t+1} = m_i\,\vec{g} + {}^{W}\vec{f}_{fromRight_i}^{t+1} - ({}^{W}\vec{\omega}_i^{t+1} \times \vec{r}_i^{t+1}) + (\frac{\vec{\tau}_{springs_i}^{t}}{I} \times \vec{r}_i^{t}). \tag{9.64} $$

9.5 Response to instantaneous forces

The effect of instantaneous external forces can be simulated by setting the initial force vector for a particular segment at a particular moment in time. For instance, to simulate the sharp tugging of a branch tip downward 10 seconds after the start of the simulation, I set the initial force ${}^{W}\vec{f}_{fromRight_i}^{t} = (0,-100,0)$ for segment i, at time $t = 10.0$ in the simulation.

9.6 Branching

Extension of this single chain of links into a branching system is straightforward. Instead of having a single child, each link can now have one or more children, each of which inherits the parent's acceleration. Because L-systems are designed to gracefully handle branching, the model's forward pass does not change: each child still has a single parent which is used as its left context. However, the backward pass must be altered, because a parent may now have more than one child from which to obtain right context. Given two or more right neighbours, the current link simply sums their values. The only values needed from the children are the unused inertial radial forces and spring forces, so each parent sums the forces passed from its children $(i+1)_R$ and $(i+1)_L$:

$$ {}^{W}\vec{f}_{fromRight_i}^{t+1} = {}^{W}\vec{f}_{fromLeft_{i+1_L}}^{t+1} + {}^{W}\vec{f}_{fromLeft_{i+1_R}}^{t+1}. \tag{9.65} $$

Equation 9.65 replaces Equation 9.57 at branching points in the model.

To model a branching structure such as a tree branch, nonzero branching angles must be specified. This translates to nonzero rest angles $\overline{\theta}$ for joint springs.

Figure 9.9 depicts such a branching system in its initial state, and in its equilibrium state as a result of torques due to gravity and spring forces.

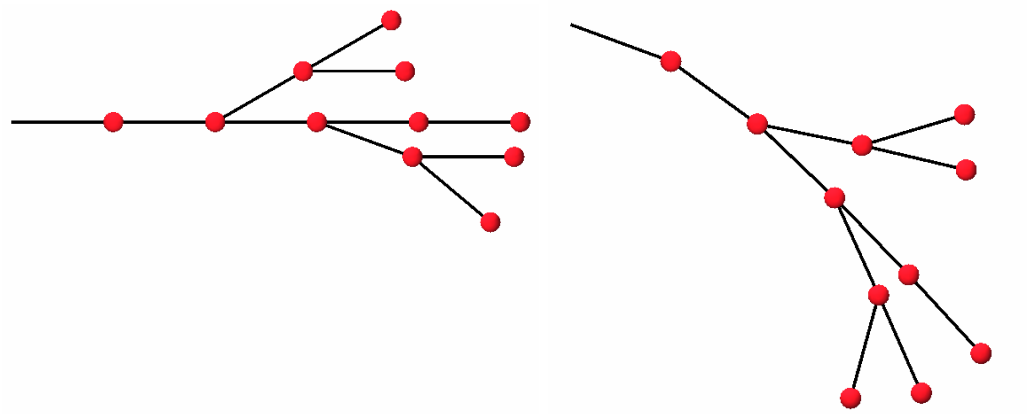


Figure 9.9: Branching rotational system in (a) its initial position and (b) its equilibrium state.

The productions for the rotational dynamics model's production rules in LPFG are given in the Appendix.

9.7 State of the solution

The solution for the rotational system presented in this chapter is adequate for small branching systems such as that of Figure 9.9. Because of the use of Euler's method for integration, this solution is not appropriate for larger systems with many segments, varying weights, larger spring constants, or initial configurations in which deviation angles differ greatly from the rest angles. In such systems, forces acting on the segments can become very large and change quickly, whereas other variables change slowly (such as the accelerations at the moment between the rise and fall of a pendulum). A system of equations is called *stiff* when large differences in magnitude exist between the independent variables in the system, causing one quantity to move toward equilibrium much faster than another [36]. For stiff systems, simple numerical methods such as Euler's cannot represent the system's behaviour unless extremely small time steps are taken. To handle stiff systems, more complex numerical

methods are needed.

Future work on this problem would involve the examination of numerical methods appropriate to stiff systems, such as implicit methods or the Crank-Nicholson method, in application to the dynamics equations given in this chapter. The next step would extend the system to three dimensions using quaternions to represent the rotational values $\vec{\varepsilon}$, $\vec{\omega}$ and $\vec{\alpha}$ and to rotate the link axis $\vec{r}$. Finally, this rotational system should be incorporated into the biomechanical simulation described in Chapter 6 so that the trees produced by the biomechanical simulation can also respond to instantaneous forces.

Chapter 10

Conclusions and future work

10.1 Conclusions

In this thesis, I have presented a biomechanical model of botanical trees which incorporates simulations of several botanical phenomena. I have extended Jirasek's biomechanical model of plant axes to represent biomechanics at branching points, so as to model growing trees under the influences of gravity and tropisms. The models were expressed using L-systems, making use of the fast information transfer provided by LPFG [29] to achieve interactive rates of simulation. Because complex and realistic models can be represented efficiently, these results are made more relevant to the computer graphics community. The model's fidelity to the principles of elasticity theory was tested using Euler's equations for the buckling length of a column, and good results were obtained.

Another contribution of this thesis is a botanically-based simulation of reaction wood. The most important characteristics of reaction wood described by botanists [33, 53] were incorporated in the model. This simulation can be used to increase resistance to bending in a leaning branch in the model, and can even accomplish correction of the lean.

The use of concepts from elasticity theory, realistic material properties and biomechanics calculations predisposes the model to handle testing for branch breakage. Branch breakage in the models occurs as a result of excessive stresses from weight, such as is experienced by some fruit trees.

Two approaches to controlling tree architecture were presented in this thesis. The function-based approach to architecture control can be used to recreate observable phenomena such as acrotony. The resource allocation strategy successfully controls which portions of the

tree will become vigourous. Used in combination with the breakage simulation, it causes increased primary growth in the year following substantial branch breakage.

The result of these contributions is a controllable tree model which behaves realistically on a biomechanical level and is based in botanical observations. By simulating the botanical and biomechanical processes of the tree's development, I produce branch shapes of similar character to those seen in nature.

As a case study, the biomechanical model was used along with the resource allocation simulation to model the crooked poplar tree. This tree demonstrates an unusually curved and twisted architecture as a result of severe bending due to gravity. The model simulates both the wild type poplar and the crooked mutant, allowing conversion between them by altering a few parameter values. This simplicity of conversion between trees of very different architectures demonstrates the robustness of the model, as well as its ability to allow architectural characteristics to occur as emergent properties of the simulation.

This thesis also produces steps toward a simulation of dynamics in tree branches. The dynamics of small branching systems under the influences of gravity and instantaneous external forces are represented. The simulation produces believable motion for small branching systems in two dimension.

10.2 Future work

Future work should extend the dynamics system to three dimensions and to larger systems. In particular, to represent the dynamics of larger structures such as entire trees will require the application of more stable numerical methods to this problem. Finally, the dynamics solution should be incorporated into the existing biomechanical tree model. This would create a powerful physically-based tree model which responds to external forces after it has grown.

An appropriate interface could be designed by which to interact with the physical sim-

ulation. By means of this interface, the user could set the direction of tropisms and specify which forces to apply to the branches.

It would also be interesting to investigate multi-resolution approaches relating to this tree model. By scaling down the complexity of the model (reducing the number of segments that represent a plant stem), the biomechanics calculations would be simplified automatically. A chain of segments in the high-resolution model could be replaced by a single segment that lies in the direction of the secant to the original chain. To move from low resolution to high resolution, a single segment could be subdivided into several smaller segments using a common curvature. These transformations to lower or higher resolution could be accomplished by production rules in the L-system. The ability to change between levels of detail quickly would be very useful in practical applications, such as when the distance from the tree to the viewer is changed.

Appendix A

Appendix: Code for the simulations

This Appendix provides LPFG code for the most important parts of the models described in this thesis. In Section A.1 the production rules for the biomechanical model including simulations of resource allocation, reaction wood and breakage are given. Section A.2 presents alterations to these production rules needed to simulate the crooked poplar tree. Section A.3 shows how the function-based architecture control strategy is implemented. Finally, the LPFG implementation of the dynamics solution is presented in Section A.4.

A.1 General biomechanical model

The following LPFG code is the complete implementation of the biomechanical model including resource allocation, reaction wood and breakage simulations.

```
#include <memory.h>
#include <lpfgall.h>
#include <stdmods.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include "vectorlib.h"

enum RotationType { QUATERNIONS, MATRICES };
RotationType rotType = QUATERNIONS;

enum Orientation { VERTICAL, DIAGONAL };
Orientation orient = VERTICAL;

#define RADTODEG 57.2957795786
#define LEAVES 1                /* whether there are leaves */
#define LEN0 0.055000           /* length of first internode */
#define RADIUS0 0.002000        /* cross-sectional radius of first internode */
#define TROP 0.000800           /* magnitude of negative geotropism */
#define GRAV -9.8               /* magnitude of gravity */
#define PIPE 2.000000           /* exponent for the pipe model */
#define SPIRAL 2.39998          /* angle for spiral phyllotaxis (137.5 in radians) */
#define MAXBUDS 21              /* maximum number of buds in a shoot's year's growth */
```

```
#define ALPHA 0.100000          /* controls speed of convergence to a solution */
#define REACT_ALPHA 0.300000
#define TRUNK_ONLY 0            /* whether reaction wood is present only on trunk, or in all branches */
#define RADIUS_GOAL 0.6         /* percentage of radius growth goal which is achieved at each timestep */
#define FRUIT_GOAL  0.9         /* percentage of fruit/leaf growth goal achieved every timestep */
#define DENSITY 77.900000       /* density of the wood */
#define LEAF_MASS 0.000002      /* mass of each leaf */
#define LEAF_DENSITY 0.095      /* density of a leaf, to determine its size from its mass */
#define FRUIT_MASS 0.010000     /* mass of a fruit */
#define FRUIT_DENSITY 300.0     /* density of a fruit */
#define FRUIT_PROB 0.270000     /* probability of producing a fruit with any bud */
#define BUDLENGTH 0.008000      /* length of internodes which are buds */
#define SUMMERLENGTH MAXBUDS    /* length of summer season */
#define E_JUV 600000.015157     /* average Young's modulus for young wood */
#define E_ADULT 158700004.0     /* Young's modulus for adult wood */
#define E_REACTION(juv) (0.5*juv + 0.5*E_ADULT)    /* reaction wood's E value */
#define G(e) (2.0*e)            /* polar moment of area, G, as a function of current E value*/
#define GETS_ADULT 5            /* age, in months, at which wood begins to age -- E starts to increase */
#define ADULT 24                /* age, in months, at which wood has achieved adulthood */
                                /* an increment for wood stiffness, each month */
#define E_INCREMENT(juv)  ((AGE_INCREMENT/12.0) * (E_ADULT - juv)/(ADULT-GETS_ADULT))

#define COLOUR 0                /* indicate wood age in colour of display */

#define TROP_DIR_X 0.000000     /* tropism direction vector (usually UP for negative geotropism) */
#define TROP_DIR_Y 1.000000
#define TROP_DIR_Z 0.000000

#define MOR 289999.986226       /* modulus of rupture for bending */
#define RW_INCREMENT 0.001      /* increment for the further ovalization of the branch due to reaction wood */
#define DEVIATION_FROM_EP 0.3   /* maximum acceptable deviation from "equilibrium position" before reaction wood production starts */

#define EQUILIBRIUM_EPSILON 0.1 /* threshhold for change in Omega at which we claim equilibrium is reached for current state */
#define EPSILON 0.00000001      /* threshhold for change in radius at which we claim radius stopped increasing */

#define TH1 0                   /* deviation angle for first child (continuation of main branch) */

#define END_YEAR 10             /* stop simulation (both growth and shedding) at this year */
#define YEARS_TO_SHED 999993    /* if a branch has had low vigour for this many years (while young), shed it */
#define SHOOT_VIGOUR .03        /* amount of resources required to start a new shoot */
#define FAIRNESS 1.000000       /* 0 means unfair distribution of resources (winner takes all) 1 means fair (according to girth) */
#define SEED 1484127            /* seed for random number generator */

// -------- MACROS AND FUNCTIONS ----------------------------------------

#define RAND(lo, hi)        (lo + (float)(hi-lo)*( (float)(rand() % 100) / 100.0 ))
#define CLAMP_PH(num)       ((num>2*M_PI) ?(num-(2*M_PI)) : (num))      /* clamp phyllotactic angle */

inline double Iellipse_big(double rBig, double rSmall)      {return M_PI*rSmall*pow(rBig,3)/4.0;}
inline double Iellipse_small(double rBig, double rSmall)    {return M_PI*rBig*pow(rSmall,3)/4.0;}
inline double Jellipse(double rBig, double rSmall)      {return M_PI*pow(rBig,3)*pow(rSmall,3) / (pow(rBig, 2) + pow(rSmall,2));}

inline double Iring_big(double rObig, double rOsmall, double rIbig, double rIsmall)     \\
            {return (M_PI* (rOsmall*pow(rObig,3)) - (rIsmall*pow(rIbig,3)))/4.0;}
```

```
inline double Iring_small(double rObig, double rOsmall, double rIbig, double rIsmall)   \\
            {return (M_PI* (rObig*pow(rOsmall,3)) - (rIbig*pow(rIsmall,3)))/4.0;}
inline double Jring(double rObig, double rOsmall, double rIbig, double rIsmall)         \\
            {return (M_PI * pow(rObig,3)*pow(rOsmall,3)) / (pow(rObig, 2) + pow(rOsmall,2)) \\
             - ( M_PI * pow(rIbig,3)*pow(rIsmall,3)) / (pow(rIbig, 2) + pow(rIsmall,2));}


/* increase in age, in months, every time 1ary/2dary growth occurs */
const float AGE_INCREMENT = (12.0/(2.8*(float)SUMMERLENGTH));

const V3f Gravity = V3f(0,GRAV,0);
const V3f up = V3f(0,1,0);  /* world up vector */
V3f Tropism = TROP * Normalize(V3f(TROP_DIR_X, TROP_DIR_Y, TROP_DIR_Z));

int year = 0;               /* for year display */
double boxsize;             /* size of the bounding box */
int dir;                    /* scanning direction */
int noSecGrowth;            /* if we are still determining equilibrium for the current state, do not do secondary growth */
bool summer = true;         /* summer has primary and secondary growth, winter just has 2ndary (radial) */
bool switchSeasons = true;  /* whether to switch seasons -- summer->winter->spring */
float vigMainBranch = 0;    /* resources in main branch when determining fate of buds in spring */

int global_countSB = 0;     /* count the number of start braces I removed, remove same number of end braces */


/* Determine whether equilibrium has been reached for current structure */
bool EquilibriumTest( float biggestChange )
{
    return ( biggestChange < EQUILIBRIUM_EPSILON );
}

/* Heuristic: calculate proportion of parent's resources that child2 gets. */
float VigourProportion( const V3f& hParent, const V3f& hChild, float areaParent, float areaChild, float height )
{
    float Q = 1.0;
    float orientation = 2*(1 - fabs(DotProd(hParent, up) )) * DotProd(hChild, up);

    float proportion = pow(areaChild, (float)PIPE/2.f) / pow(areaParent, (float)PIPE/2.f);
    proportion *= height;

    /* I have found "fair" proportion: now introduce unfairness.
    * This is "winner takes all" way. fraction of fair for smaller proportion, 1-smallerproportion for larger proportion */
    if( proportion > 0.5 )         proportion = ( 1.0 - FAIRNESS*(1-proportion) );
    else if( proportion < 0.5 )    proportion = FAIRNESS*proportion;

    // Clamp at 1
    if( proportion > 1 )            proportion = 1;

    // Only distribute a certain percentage of resources this year (save some to extend to next year)
    return proportion*Q;
}

/* Determine whether to make a bud into a shoot, based on resources only. */
bool BudShoots( float vigour )
```

```
{
    if( vigour > SHOOT_VIGOUR ) return true;
    else return false;
}


/* Determine how many buds to give to the child shoot. Depends on amount of resources. */
int ChildNumBuds( float vigour )
{
    int result = (func(NUMBUDS_VIGOUR, (vigour)) * MAXBUDS) ;
    return (result>=1)? result : 1;
}


/* Reorient axis of child according to its branching angles th, ph.
 * There are two hlu frames: phylloHlu takes care of phyllotaxis, hlu keeps hlu.U closest to up. */
void Reorient( TurtleVec& phylloHlu, TurtleVec& hlu, float th, float ph, V3f Omega, float length )
{
    V3f N = phylloHlu.L;
    switch( rotType ) {
        case(QUATERNIONS) :
        {
            //Spiral phyllotaxis: rotate around the H vector
            Quaternion h = MakeQuaternion(phylloHlu.H, ph);
            N = h * N;

            // Now make a shoot off at angle th, and update both phylloHlu and other hlu.
            Quaternion q = MakeQuaternion(N, th);
            phylloHlu = q * phylloHlu;
            break;
        }
        case(MATRICES):
        {
            // Spiral phyllotaxis: rotate around the H vector
            N = VecRotate( N, phylloHlu.H, ph );

            // Now make a shoot off at angle th, and update phylloHlu.
            VecRotate( phylloHlu, N, th );
            break;
        }
        default: break;
    }

    hlu.H = phylloHlu.H;

    float angle = Magnitude(Omega) * length;
    V3f Omega_unit = Normalize(Omega);

    switch( rotType ) {
        case(QUATERNIONS) :
        {
            Quaternion q = MakeQuaternion( Omega_unit, angle );
            hlu.H = q * hlu.H;
            phylloHlu = q * phylloHlu;
```

```
            break;
        }
        case(MATRICES) :
        {
            VecRotate( hlu.H, Omega_unit, angle);
            VecRotate( phylloHlu, Omega_unit, angle);
            break;
        }
        default: break;
    }

    /* Reorient the HLU frame so that U axis is also the semi-axis of the cross-section which lies in the direction of bending.
    * (ie: the semi-axis to be lengthened if reaction wood is produced)
    * Note: Assumes that plane of bending is always XZ (bending DOWN or UP only).
    */

    V3f crossprod = V3f(0,1,0) % hlu.H;
    if( Magnitude(crossprod) > 0.01 )
    {
        if( DotProd(hlu.H % crossprod, hlu.U) > 0 )
        {
            hlu.U = hlu.H % (crossprod);
        }
        else
        {
            // If dot product is negative, it means there will be a "twist" since H crossed over 010.
            // So keep it from twisting.
            hlu.U = -1.0 * (hlu.H % (crossprod));
        }
        hlu.U.Normalize();
        hlu.L = hlu.U % hlu.H; //right hand rule
        hlu.L.Normalize();
    }

}


// ------ STRUCTURES -------------------------------------------------

/* Naming conventions for members: uppercase means Global, lowercase means local */
struct InternodeData
{
    double mass;         /* mass at the joint */
    double sigmaMass;    /* combined mass to the right of the joint */
    double len;          /* internode length */
    double rSmall, rBig; /* smaller and larger semi-axes of the ellipse */
    double goal_rBig, goal_rSmall; /* goal for radius of cross-section, according to pipe model.
                                    * This goal will be reached assymptotically over time, simulating need for resources */
    TurtleVec HLU;       /* turtle heading, left, up */
    TurtleVec phylloHLU; /* turtle used for phyllotaxis only, so it doesn't interfere with orientation of cross-section for reaction wood, rigidity, etc */
    V3f Omega;           /* incremental turtle rotation */
    V3f Omega_e;         /* rest angle at the joint */
    V3f AccumTorque;     /* accumulated torque for segments to right */
    LocVec rigidity;     /* flexural and torsional rigidities */
```

```
    float phyllo;        /* angle around H vector for phyllotaxis */
    float mi;            /* mass increment */
    float changeInOmega; /* diff between current Omega and previous Omega; to see when we reach Equilibrium */
    float ageMonth;      /* rough age of this segment in MONTHS (heuristic, may not always be accurate to age in years), used to calculate E */
    int ageYear;         /* age in years, incremented in spring */
    int order;           /* order of branching (0 for trunk). Used if putting reaction wood on trunk */
    float vigour;        /* vigour (amount of resources) in this internode */
    int lowVigStart;     /* age when I started having low vigour. If stays low vigour for YEARS_TO_SHED years, shed me */
    float height;        /* height in year (as a percentage of numBuds), used in determining amount of resources to allocate*/
    int count;           /* which bud in year's growth? */
    int which;           /* main==1 or lateral==2 */
    bool isBud;          /* is it a bud? */
    bool oneChild;       /* has only one child -- so only child inherits all resources */
    bool takesVigour;    /* if this is a bud scar, take no resources */
    float vig2;          /* resources of my lateral child. Continuing main child will need to know it */
    bool startOfBuddingBranch;   /* Whether this internode is the start of a budding branch -- for fate of buds in spring */
    V3f Pos;             /* position in world space -- to stop growing when hits ground */
    int juvE;            /* stochastic value of juvenile wood's Young's modulus (varies with each shoot) */
    bool isBase;         /* is this the base of the tree -- only the base does equilibrium test in bkwd direction */
};


// ----- MODULES ------------------------------------------------------

module Internode(InternodeData);        /* An internode */
module BranchAngle(float, int, float);  /* Deviation angle before a lateral branch*/
module Apex(int, int, int);             /* The apex */
module PotentialApex(bool, int);        /* Might turn into an apex */
module Leaf(float, float, TurtleVec);   /* Leaf, exists for only a year (falls off in winter) */
module Fruit(float, float);             /* Fruit, as above */
module Box();                           /* The bounding box */
module Scar();                          /* The end of a pruned branch. Needed to trigger removal of any child internodes, leaves etc. */

// ----- GROUPS (use scanning direction) ------------------------------

#define FWD     1
#define BKWD        2
#define INCREMENT_YR    3

InternodeData nid, firstid;     /* new internode data used for all new internodes, and first internode respectively */

// ----- INITIALIZATIONS ----------------------------------------------

Start:
{
    srand(SEED);
    iset = 0;     /* used in random number generation */
    nid.ageMonth = nid.ageYear = 0;
    nid.sigmaMass = 0;
    nid.len = LEN0;
    nid.rBig = nid.rSmall = nid.goal_rBig = nid.goal_rSmall = RADIUS0;
    nid.mass = DENSITY*nid.len*M_PI*nid.rBig*nid.rSmall;

    if( orient == VERTICAL )
```

```
    {
        /* Orientation of Tree */
        nid.HLU.H = V3f(0, 1, 0);
        nid.HLU.L = V3f(0, 0, -1);
        nid.HLU.U = V3f(1, 0, 0);
    }
    else
    {
        /* Orientation of Branch */
        nid.HLU.H = V3f(0.707,0.707,0);
        nid.HLU.L = V3f(0,0,-1);
        nid.HLU.U = V3f(-0.707,0.707,0);
        nid.HLU = Normalize(nid.HLU);
    }

    nid.phylloHLU = nid.HLU;

    nid.Omega = V3f(0,0,0);
    nid.Omega_e = V3f(0,0,0);

    nid.rigidity = LocVec(  G(E_JUV)*Jellipse(nid.rBig, nid.rSmall),
                            E_JUV*Iellipse_small(nid.rBig, nid.rSmall),
                            E_JUV*Iellipse_big(nid.rBig, nid.rSmall));    //rigidity around each of local h,l,u axes

    nid.AccumTorque = V3f(0,0,0);
    nid.phyllo = 0;
    nid.changeInOmega = 0;
    nid.lowVigStart = -1;
    nid.isBud = false;
    nid.order = 0;
    nid.takesVigour = true;
    nid.oneChild = false;
    nid.vig2 = 0;
    nid.startOfBuddingBranch = false;
    nid.Pos = V3f(0,0,0);
    nid.juvE = E_JUV;
    nid.isBase = false;

    firstid = nid;
    firstid.isBase = true;
    firstid.startOfBuddingBranch = true;
    firstid.Pos = nid.Pos + firstid.HLU.H*firstid.len;

    Tropism = TROP * Normalize(V3f(TROP_DIR_X, TROP_DIR_Y, TROP_DIR_Z));
    boxsize = 20*LEN0;    /* size of bounding box for displaying */
    year = 0;
    summer = true;
    dir = FWD;
    Forward();
    UseGroup( dir );

    global_countSB = 0;
}
```

```
StartEach:
{
    UseGroup( dir );
    if( dir == BKWD ) Backward();
    else {
        noSecGrowth=false;
        Forward();
    }
}

EndEach:
{
    if( year > END_YEAR ) dir = 0;

    else if( switchSeasons && dir!= FWD )
    {
        if( dir==INCREMENT_YR ) //spring
        {
            year++;
            dir = FWD;
            // will be summer next.
        }
        else if( summer )
        {
            summer = false;
            dir = FWD;
        }
        else //winter
        {
            summer = true;
            dir = INCREMENT_YR;
        }
    }
    else
    {
        // Within a season, alternate between fwd and bkwd
        dir = 3 - dir;
        }

    if( dir != BKWD ) switchSeasons = true;

}

derivation length: 1;

// -------- PRODUCTIONS ----------------------------------------------

Axiom:  SB Box EB
    SetColor(9)
    StartGC() SetWidth(RADIUS0)
    Internode(firstid)
    Apex(0, MAXBUDS/2.0, 0) EndGC();
```

```
group INCREMENT_YR: //-------------------------------------------------------------------------

/* PotentialApex can turn into an apex at next year. Probability of doing so
* is based on amount of resources, which are propagated from base of tree, and modulated by height.
*/
Internode(idl) << BranchAngle(ph, which, th) Internode(id) PotentialApex(leafed, apexAge) :
{
        id.ageYear++;

        /* Determine vigour (amount of resources). Because of order of string interpretation, subtract from a global variable.
        * Set global vigMainBranch if i am the start of the branch */
        if( idl.startOfBuddingBranch )
            vigMainBranch = idl.vigour;

        id.vigour = vigMainBranch * VigourProportion(idl.HLU.H, id.HLU.H, M_PI*idl.goal_rBig*idl.goal_rSmall,
                        M_PI*id.goal_rBig*id.goal_rSmall, id.height);

        /* If the bud shoots, create a sink which removes resources needed to shoot */
        if( BudShoots(id.vigour) )
        {
            float sinkFactor = ChildNumBuds(id.vigour)/4.0;
            vigMainBranch -= id.vigour * sinkFactor;

            /* Sanity check */
            if( vigMainBranch < 0 ) vigMainBranch = 0;

            id.isBud = false;
            id.startOfBuddingBranch = true;
            produce BranchAngle(ph, which, th) Internode(id)
                Apex(0, ChildNumBuds(id.vigour), 0);
        }
        else
        {
            id.takesVigour = false;
            produce BranchAngle(ph, which, th) Internode(id);
        }
}


/* Reset count to 0 for the new year */
Apex(count, numBuds, age) :
{
    count = 0;
    produce Apex(count, numBuds, age);
}

Internode(id) :
{
    id.ageYear++;
    produce Internode(id);

}
group FWD: // ------------------ * Forward pass * ----------------------------------------------
```

```
Internode(id) << Apex(count, numBuds, age) :
{
    /* Apex produces a new internode once equilibrium is reached for the current structure. */

    if ( !EquilibriumTest( id.changeInOmega ) )
    {
        noSecGrowth = true;
        switchSeasons = false;  // Do not switch seasons if equilibrium not reached for current state
        produce Apex(count, numBuds, age) ;
    }

    /* Do not go on unless equilibrium has been reached -- if so: In winter, make only secondary growth. In summer, produce a bifurcation.
    * Branch at regular intervals phyllotactically, creating a continuation of main branch
    * and a new bud which may or may not produce a shoot. */

    else if( !noSecGrowth )
    {
        /* In winter, do not grow new segments, but still adjust */
        if( !summer )
        {
            produce Apex(count, numBuds, age);
        }
        /* If I hit the ground, stop growing */
        else if( id.Pos.y < BUDLENGTH )
        {   produce Scar;
        }
        else
        {
            switchSeasons = false;  // Do not switch seasons until all intended new buds have been formed
            count++;

            /* Pass on info from parent */
            InternodeData id1 = nid, id2 = nid;
            id1.phyllo = CLAMP_PH(id.phyllo + SPIRAL);
            id2.phyllo = 0;     // Restart phyllotaxis angle from zero if there is a new shoot

            id1.HLU = id2.HLU = id.HLU;
            id1.phylloHLU = id2.phylloHLU = id.phylloHLU;

            id2.len = BUDLENGTH;
            id2.isBud = true;

            id1.order = id.order;
            id2.order = id.order + 1;

            id1.which = 1;
            id2.which = 2;

            /* height in year is used in determining vigour */
            id1.height = id2.height = (float)count/(float)numBuds;
            id1.count = id2.count = count;

            /* Stochastically assign juvE value -- it varies a little in each shoot */
            id1.juvE = id.juvE;
```

```
id2.juvE = nrand(E_JUV, E_JUV/6.0);

/* Angle of potential new shoot (smaller for wild type) */
float th2 = RAND( func(ANGLE2_MIN_HEIGHT, id2.height)*100.0/RADTODEG, func(ANGLE2_MAX_HEIGHT, id2.height )*100.0/RADTODEG );

/* Add tropism for each child branch. Affects only initial Omega_e, Omega.
* Before calculating, must reorient reference frame of children
* according to phyllotaxis and deviation angles. */

Reorient(id1.phylloHLU, id.HLU, TH1, id.phyllo, id1.Omega, id.len);
Reorient(id2.phylloHLU, id.HLU, th2, id.phyllo, id2.Omega, id.len);

V3f Torque = CrossProd(id1.HLU.H*id.len, Tropism);
// Make it local
LocVec locTorque = ToLocal(Torque, id1.HLU);
id1.Omega = id1.Omega_e = ToWorld( locTorque / id1.rigidity, id1.HLU );

Torque = CrossProd(id2.HLU.H*id.len, Tropism);
// Make it local
locTorque = ToLocal(Torque, id2.HLU);
id2.Omega = id2.Omega_e = ToWorld( locTorque / id2.rigidity, id2.HLU );

/* Update masses and lengths : mass = density * length * cross-sectional area */
id1.mass = id1.mi = DENSITY*id1.len*M_PI*id1.rBig*id1.rSmall;
id2.mass = id2.mi = DENSITY*id2.len*M_PI*id2.rBig*id2.rSmall;

id1.Pos = id.Pos + id1.HLU.H*id1.len;
id2.Pos = id.Pos + id2.HLU.H*id2.len;

if( count < numBuds) {
    produce SB() // Right Branch -- small potential shoot
        BranchAngle(id2.phyllo, 2, th2)
        Internode(id2)
        PotentialApex(false, 0) EB()

        SB() // Continuation of main
        BranchAngle(id1.phyllo, 1, TH1)
        Internode(id1)
        Apex(count, numBuds, age)
        EB();
}
else    /* The end of this year's main branch becomes a bud (potentially continues growing) for next year */
{
        // Increment the age of this apex
    age++;
    produce SB() // Right Branch -- small potential shoot
        BranchAngle(id2.phyllo, 2, th2)
        Internode(id2)
        PotentialApex(false, 0) EB()

        SB() // Bud for potential continuation of main
        BranchAngle(id1.phyllo, 1, TH1)
        Internode(id1)
        PotentialApex(false, age)
```

```
                    EB();
            }
        }
    }
}


/* Buds may have leaves and possibly a fruit.*/
Internode(idl) BranchAngle(ph, which, th) Internode(id) << PotentialApex(leafed, apexAge) :
{
    if( LEAVES )
    {
        if( !leafed )
        {
            float goal_leaf_mass = nrand(LEAF_MASS, LEAF_MASS/2.0);

            // Fruit created probibilistically, leaves created every time (if leaves specified at all).
            float fruit = RAND( 0.0, 2.0*(MAXBUDS -id.count)/MAXBUDS );
            if( fruit < FRUIT_PROB  && id.count > 2 )
            {
                float goal_fruit_mass = nrand(FRUIT_MASS, FRUIT_MASS/2.0);

                produce PotentialApex(true, apexAge)
                    Leaf(goal_leaf_mass/3.0f, goal_leaf_mass, id.HLU)
                    Fruit(goal_fruit_mass/3.0f, goal_fruit_mass);
            }
            else
                produce PotentialApex(true, apexAge)
                    Leaf(goal_leaf_mass/3.0f, goal_leaf_mass, id.HLU);
        }
    }

}

/* Branched front propagation */
Internode(idl) << BranchAngle(ph, which, th) Internode(id) :
{
    // Start with left context's orientation
    id.phylloHLU = idl.phylloHLU;
    id.HLU = idl.HLU;

    /* Apply branching angle first.
    * Use deviation angle for appropriate child branch */
    Reorient(id.phylloHLU, id.HLU, th, idl.phyllo, id.Omega, idl.len);

    id.Pos = idl.Pos + id.HLU.H*id.len;

    /* If I pass through the ground by bending, crop */
    if( id.Pos.y < -1*BUDLENGTH )
        produce Scar;

    /* Determine portion of parent's resources that I get. If I'm a bud scar, don't take any and all goes to my sister.
    * If I'm an only child, take all. Otherwise share. */
```

```
    if( !id.takesVigour )
        id.vigour = 0;
    else if( idl.oneChild )
        id.vigour = idl.vigour;
    else
    {

        /* Give every bud in the year a roughly even amount of resources. To do so, use radii according to the pipe model.
        * Ensure that resources of two children equals incoming resources. */

        if( which==2 ) //bud
        {
            id.vigour = idl.vigour * VigourProportion(idl.HLU.H, id.HLU.H, M_PI*idl.goal_rBig*idl.goal_rSmall,
                        M_PI*id.goal_rBig*id.goal_rSmall, id.height);
        }
        else
        {
            /* First (continuation) child. Just take remaining resources. Use vig2 that was stored at parent in the last backward pass. */
            id.vigour = idl.vigour - idl.vig2;
            // Sanity check -- Clamp at 0 -- in case old vigour 2 was more than it currently is
            if( id.vigour < 0 ) id.vigour = 0;
        }
    }

    /* Shed a branch if it hasn't grow radially for too many years. */
    if( id.goal_rBig - id.rBig < EPSILON )
    {
        // If this is new, start counting amount of time at low vigour
        if( id.lowVigStart == -1 ) {
            id.lowVigStart = id.ageYear;
        }
        else if( id.ageYear - id.lowVigStart >= YEARS_TO_SHED )
            produce Scar;
    }
    // If vigour level is ok, reset the countdown to counteract any impending death
    else id.lowVigStart = -1;

    // Send biggest changeInOmega result (which was found in backpass) forward through chain
    id.changeInOmega = idl.changeInOmega;

    produce BranchAngle(ph, which, th) Internode(id);
}

/* The very first internode (the base of the tree trunk) */
Internode(baseId) :
{
    /* Base of trunk decides the quantity of resources to propagate through entire tree -- determined by radius of base */
    baseId.vigour = pow(baseId.rBig*800, 1.1);

    if( !EquilibriumTest( baseId.changeInOmega ) )
    {
        noSecGrowth = true;     // No more growth until I reach equilibrium
        switchSeasons = false;  // Do not switch seasons if equilibrium not reached for current state
    }
```

```
    produce Internode(baseId);
}


/* Count how many SBs I remove after each scar, so I can remove the same amount of EBs */
Scar() << SB() :
{
    global_countSB++;
    produce;

}

/* Eliminate all subsequent internodes after a scar */
Scar() << BranchAngle(ph, which, th) Internode(id) :
{   produce;
}

/* Eliminate PotentialApex after a scar */
Scar() << PotentialApex(leafed, apexAge):
{
    produce;
}

/* Eliminate Leaf after a scar */
Scar() <<  Leaf(mass, goal_mass, hlu):
{
    produce;
}

/* Eliminate Fruit after a scar */
Scar() <<  Fruit(mass, goal_mass):
{
    produce;
}

/* Eliminate Apex after a scar */
Scar() << Apex(count, numBuds, age) :
{
    produce;
}

/* Remove the same number of EBs as SBs that were removed */
Scar() << EB() :
{
    if( global_countSB > 0 )
    {
        global_countSB--;
        produce;
    }
}

Leaf(mass, goal_mass, hlu) :
{
    /* Leaves only exist for one year, are shed in winter.
```

```
    * Shed them slowly (stochastically) within winter, not all in same timestep. */
    if( !summer && nrand(1.0, 0.8) < 1.0 ) produce;
}

Fruit(mass, goal_mass) :
{
    /* Fruit only exists for one year, is shed in winter.
    * Shed them slowly (stochastically) within winter, not all in same timestep. */
    if( !summer && nrand(1.0, 0.5) < 1.0 ) produce;
}

group BKWD: // ---------------- * Backward pass * ----------------------------------------------------

/* Remove branch angle before a scar */
BranchAngle(a, b, c) >> Scar():
{
    produce;
}

/* Remove scars, because they get in the way of matching other production rules */
Scar() :
{
    produce;
}
/* Also get rid of the empty branches they leave */
SB() EB() :
{
    produce;
}

/* Increase the mass of fruit every time equilibrium is reached */
Fruit(current_mass, goal_mass) :
{
    if(!noSecGrowth)
        produce Fruit(current_mass + (goal_mass - current_mass)*FRUIT_GOAL, goal_mass);
}

/* Increase the mass (and size) of leaves every time equilibrium is reached */
Leaf(current_mass, goal_mass, hlu):
{
    if(!noSecGrowth)
        produce Leaf(current_mass + (goal_mass - current_mass)*FRUIT_GOAL, goal_mass, hlu);
}


/* 2-Branched back-propagation. Note: Child 2 (the lateral bud or branch) is now first in the string. */
Internode(id) >> SB() BranchAngle(ph2, which2, th2) Internode(id2) /*...*/ EB()
        SB() BranchAngle(ph1, which1, th1) Internode(id1) /*...*/ :
{
    if( !id1.takesVigour || !id2.takesVigour )
        id.oneChild = true;

    id.vig2 = id2.vigour;   // Store this because child1 will need to know it.
```

```
LocVec locOmega, locOmega_e, locNewOmega_e, newRigid;

/* change system parameters due to secondary growth */
    if (!noSecGrowth) {
    id.ageMonth += AGE_INCREMENT;

    id.mi = id1.mi + id2.mi;
    id.mass += id.mi;

    // update 3 rigidity values of aging inner branch -- keep existing rigidity, but increment it
    LocVec I_J_current = LocVec(Jellipse(id.rBig, id.rSmall), Iellipse_small(id.rBig, id.rSmall), Iellipse_big(id.rBig, id.rSmall));
    LocVec E_G_current = id.rigidity / I_J_current;

    // Increment the rigidity by the age increment, if I am not already as stiff as adult wood gets
    if( E_G_current.l < E_ADULT )
        E_G_current = E_G_current + LocVec(E_INCREMENT(id.juvE),E_INCREMENT(id.juvE),E_INCREMENT(id.juvE));

    id.rigidity = E_G_current * I_J_current;

    float oldRBig = id.rBig;
    float oldRSmall = id.rSmall;

    /* calculate new goals for ellipse semi-axes, keeping their existing length ratio constant.
    * Do this by calculating area and then solving for each of rSmall, rBig using ratio. */
    float ratio = id.rSmall/id.rBig;

    /* Area cannot be reduced (even if a branch was shed further up), so use Max with current area */
    float areaParent = Max( M_PI*id.rBig*id.rSmall, pow( pow(M_PI*id1.rBig*id1.rSmall, PIPE/2.0)
                        + pow(M_PI*id2.rBig*id2.rSmall, PIPE/2.0) ,(1.f/(PIPE/2.0))));

    // This comes from areaParent = M_PI*rBig*rSmall = M_PI*rBig*(ratio*rBig)
    id.goal_rBig = sqrt( areaParent/(ratio*M_PI) );
    id.goal_rSmall = ratio*id.goal_rBig;

    /* Check whether reaction wood is needed */
    float E_ring = id.juvE;         // this will change below if there is reaction wood
    V3f Ring_omega_e = id.Omega;    // new ring's rest angle -- if reaction wood, will be shifted away from current lean

    if( REACT_ALPHA > 0.0 && (!TRUNK_ONLY || ( TRUNK_ONLY && id.order == 0 ))
        && ( Magnitude(id.Omega - id.Omega_e) > DEVIATION_FROM_EP ))
    {
        id.goal_rBig+= RW_INCREMENT;

        E_ring = E_REACTION(id.juvE);
        Ring_omega_e = id.Omega*(1.0-REACT_ALPHA) + id.Omega_e*(REACT_ALPHA);
    }

    id.rBig  += (id.goal_rBig - id.rBig)*RADIUS_GOAL;
    id.rSmall += (id.goal_rSmall - id.rSmall)*RADIUS_GOAL;


    /* Only add a new layer if r increased */
    if( id.rBig - oldRBig > EPSILON/1.0 )
    {
```

```
        // Winter should last until secondary growth has caught up
        if( !summer ) switchSeasons = false;

        // rigidity of the newly added ring
        newRigid = LocVec(  G(E_ring) * Jring(id.rBig, id.rSmall, oldRBig, oldRSmall),
                    E_ring * Iring_small(id.rBig, id.rSmall, oldRBig, oldRSmall),
                    E_ring * Iring_big(id.rBig, id.rSmall, oldRBig, oldRSmall));

        LocVec localomega_e = (id.rigidity * ToLocal(id.Omega_e, id.HLU) +
            newRigid * ToLocal(Ring_omega_e, id.HLU) ) / (id.rigidity + newRigid);

        id.Omega_e = ToWorld(localomega_e, id.HLU);

        // calculate new total rigidity
        id.rigidity = id.rigidity + newRigid;
    }
}

V3f Torque, Torque1, Torque2;
V3f NewOmega, OldOmega;
LocVec locAccumTorque;

/* accumulate mass and torque, and update joint angles */
id.sigmaMass = id1.sigmaMass + id2.sigmaMass + id.mass;
Torque1 = CrossProd(id1.HLU.H*id1.len, (Gravity * id1.sigmaMass));
Torque2 = CrossProd(id2.HLU.H*id2.len, (Gravity * id2.sigmaMass));
Torque = Torque1 + Torque2;
id.AccumTorque = id1.AccumTorque + id2.AccumTorque + Torque;
locAccumTorque = ToLocal(id.AccumTorque, id.HLU);


/* Test for breakage */
LocVec stress = locAccumTorque*id.rBig / (double)Iellipse_big(id.rBig, id.rSmall);
if( Magnitude(stress) > MOR )
{
    // modulus of rupture exceeded: break here
    produce Scar;
}

// calculate new Omega using update formula
NewOmega = ToWorld( (locAccumTorque / id.rigidity), id.HLU);
OldOmega = id.Omega;
id.Omega = (id.Omega * (1.0-ALPHA)) + ((id.Omega_e + NewOmega) * ALPHA);

// Determine LARGEST change in Omega from me and all my children
id.changeInOmega = Max( Magnitude(id.Omega - OldOmega), id1.changeInOmega, id2.changeInOmega );

if( id.isBase && !EquilibriumTest(id.changeInOmega) )
{
    noSecGrowth = true;
    switchSeasons = false;
}

produce Internode(id) ;
```

```
}


/* 1-Branched back-propagation : for instance, if a Scar was made (one of pre-existing children was shed because of low vigour) */
Internode(id) >> SB() BranchAngle(ph1, which1, th1) Internode(id1) :
{
    /* Store the information that I only have one child, for resource distribution to all go to single child. */
    id.oneChild = true;
    id.vig2 = 0;

    LocVec locOmega, locOmega_e, locNewOmega_e, newRigid;

    /* Change system parameters due to secondary growth */
        if (!noSecGrowth) {
        id.ageMonth += AGE_INCREMENT;
        id.mi = id1.mi;
        id.mass += id.mi;

        // update 3 rigidity values of aging inner branch -- keep existing rigidity, but increment it
        LocVec I_J_current = LocVec(Jellipse(id.rBig, id.rSmall), Iellipse_small(id.rBig, id.rSmall), Iellipse_big(id.rBig, id.rSmall));
        LocVec E_G_current = id.rigidity / I_J_current;

        // Increment the rigidity by the age increment, if I am not already as stiff as adult wood gets
        if( E_G_current.l < E_ADULT )
            E_G_current = E_G_current + LocVec(E_INCREMENT(id.juvE),E_INCREMENT(id.juvE),E_INCREMENT(id.juvE));

        id.rigidity = E_G_current * I_J_current;

        float oldRBig = id.rBig;
        float oldRSmall = id.rSmall;

        /* Calculate new goals for ellipse semi-axes, keeping their existing length ratio constant.
        * Do this by calculating area and then solving for each of rSmall, rBig using ratio. */
        float ratio = id.rSmall/id.rBig;

        /* Area cannot be reduced (even if a branch was shed further up), so use Max with current area */
        float areaParent = Max( M_PI*id.rBig*id.rSmall, M_PI*id1.rBig*id1.rSmall );

        // This comes from areaParent = M_PI*rBig*rSmall = M_PI*rBig*(ratio*rBig)
        id.goal_rBig = sqrt( areaParent/(ratio*M_PI) );
        id.goal_rSmall = ratio*id.goal_rBig;

        /* Check whether reaction wood is needed */
        float E_ring = id.juvE;         // this will change below if there is reaction wood
        V3f Ring_omega_e = id.Omega;    // new ring's rest angle -- if reaction wood, will be shifted away from current lean

        if( REACT_ALPHA > 0.0 && (!TRUNK_ONLY || ( TRUNK_ONLY && id.order == 0 ))
            && ( Magnitude(id.Omega - id.Omega_e) > DEVIATION_FROM_EP ))
        {
            id.goal_rBig+= RW_INCREMENT;
            E_ring = E_REACTION(id.juvE);
            Ring_omega_e = id.Omega*(1.0-REACT_ALPHA) + id.Omega_e*(REACT_ALPHA);
        }
```

```
    id.rBig += (id.goal_rBig - id.rBig)*RADIUS_GOAL;
    id.rSmall += (id.goal_rSmall - id.rSmall)*RADIUS_GOAL;

    /* Only add a new layer if r increased */
    if( id.rBig - oldRBig > EPSILON/1.0 )
    {
        // Winter should last until secondary growth has caught up
        if( !summer ) switchSeasons = false;

        // rigidity of the newly added ring
        newRigid = LocVec(  G(E_ring) * Jring(id.rBig, id.rSmall, oldRBig, oldRSmall),
                    E_ring * Iring_small(id.rBig, id.rSmall, oldRBig, oldRSmall),
                    E_ring * Iring_big(id.rBig, id.rSmall, oldRBig, oldRSmall));

        LocVec localomega_e = (id.rigidity * ToLocal(id.Omega_e, id.HLU) +
            newRigid * ToLocal(Ring_omega_e, id.HLU) ) / (id.rigidity + newRigid);

        id.Omega_e = ToWorld(localomega_e, id.HLU);

        // calculate new total rigidity
        id.rigidity = id.rigidity + newRigid;
    }
}

V3f Torque, Torque1;
V3f NewOmega, OldOmega;
LocVec locAccumTorque;

/* accumulate mass and torque, and update joint angles */
id.sigmaMass = id1.sigmaMass + id.mass;
Torque1 = CrossProd(id1.HLU.H*id1.len, (Gravity * id1.sigmaMass));
Torque = Torque1;
id.AccumTorque = id1.AccumTorque + Torque;
locAccumTorque = ToLocal(id.AccumTorque, id.HLU);

/* Test for breakage */
LocVec stress = locAccumTorque*id.rBig / (double)Iellipse_big(id.rBig, id.rSmall);
if( Magnitude(stress) > MOR )
{
    // Modulus of rupture exceeded: break here
    produce Scar;
}

// Calculate new Omega using update formula
NewOmega = ToWorld( (locAccumTorque / id.rigidity), id.HLU);
OldOmega = id.Omega;
id.Omega = (id.Omega * (1.0-ALPHA)) + ((id.Omega_e + NewOmega) * ALPHA);

// Determine LARGEST change in Omega from me and all my children
id.changeInOmega = Max( Magnitude(id.Omega - OldOmega), id1.changeInOmega );

if( id.isBase && !EquilibriumTest(id.changeInOmega) )
{
    noSecGrowth = true;
```

```
        switchSeasons = false;
    }

    produce Internode(id) ;
}

/* Internode with a leaf and a fruit*/
Internode(id) >> PotentialApex(leafed, apexAge) Leaf(leaf_mass, goal_leaf_mass,  hlu) Fruit(fruit_mass, goal_fruit_mass):
{
    if (!noSecGrowth) id.ageMonth += AGE_INCREMENT;
    else id.mi = 0;

    id.sigmaMass = id.mass + leaf_mass + fruit_mass;

    id.changeInOmega = 0;
    produce Internode(id);

}

/* Internode with a leaf */
Internode(id) >> PotentialApex(leafed, apexAge) Leaf(leaf_mass, goal_leaf_mass, hlu) :
{
    if (!noSecGrowth) id.ageMonth += AGE_INCREMENT;
    else id.mi = 0;

    id.sigmaMass = id.mass + leaf_mass;
    id.changeInOmega = 0;
    produce Internode(id);

}

/* Internode with a potential apex */
Internode(id) >> PotentialApex(leafed, apexAge) :
{
    if (!noSecGrowth) id.ageMonth += AGE_INCREMENT;
    else id.mi = 0;

    id.sigmaMass = id.mass;
    id.changeInOmega = 0;
    produce Internode(id);

}

Internode(id) >> Apex(count, numBuds, age) :
{
    if (!noSecGrowth) id.ageMonth += AGE_INCREMENT;
    else id.mi = 0;

    id.sigmaMass = id.mass;
    id.changeInOmega = 0;
    produce Internode(id);

}
```

```
/* The last internode, end of a branch that will not grow any further (no Apex or PotentialApex)*/
Internode(id) :
{
    if (!noSecGrowth) id.ageMonth += AGE_INCREMENT;
    else id.mi = 0;

    id.sigmaMass = id.mass;
    id.changeInOmega = 0;
    produce Internode(id);

}
group 0: // ==========================================================================================
interpretation:


Internode(id):
{
    V3f delta;
    delta.x = id.HLU.H.x*id.len, delta.y = id.HLU.H.y*id.len, delta.z = id.HLU.H.z*id.len;

    int colour;
    if( COLOUR ) {
        if( id.ageMonth > ADULT ) colour = 1;
        else if( id.ageMonth > GETS_ADULT ) colour = 2;
        else colour = 9;
    } else {
        colour = 10;
    }

    // Colour reaction wood black
    if( id.rBig > id.rSmall )
        colour = 12;

    produce SetColor(colour)
        //Scale the contour that is drawn according to HLU frame's L (small semi-axis) and U (long semi-axis) -- makes it an ellipse
        SetHead(id.HLU.H.x, id.HLU.H.y, id.HLU.H.z,id.HLU.U.x, id.HLU.U.y, id.HLU.U.z)
        ScaleContour(2*id.rSmall, 2*id.rBig)
        F(id.len);
}


Box() :
{
    produce  f(2*boxsize);
}


Leaf(mass, goal_mass, hlu):
{
    produce SetHead(hlu.H.x, hlu.H.y, hlu.H.z, hlu.U.x, hlu.U.y, hlu.U.z)
        RollToVert()
        SetColor(9) Surface(0, pow((mass/LEAF_DENSITY), 0.5));

}
```

```
Scar():
{
    produce;
}

Fruit(mass, goal_mass):
{
    produce SetColor(4) Sphere( pow( (0.75/M_PI)*(mass/FRUIT_DENSITY), 0.3333));
}

YearLabel():
{
    char* yearLabel = new char[15];
    sprintf(yearLabel, "Year: %d", year);
    produce f(.1) Label( yearLabel ) f(.1);

}
```

Many of the model's parameters, found in the `#define` section of the code above, can be set by the user. Figure A.1 shows the Panels tab in L-studio, by which these parameters can be altered.

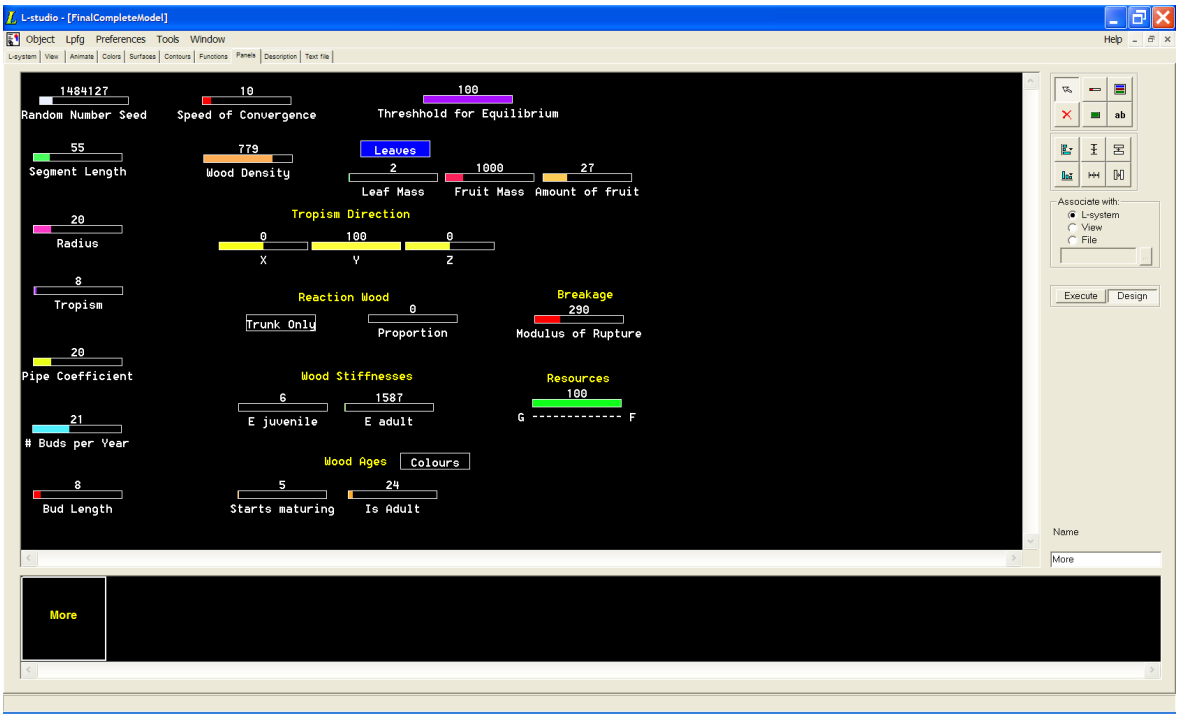


Figure A.1: Panels allow user control over the L-system's parameters

A.2 Crooked poplar

Several alterations are needed to the above model to represent the crooked and wild type poplar tree. Most importantly, the stiffness E of wood must vary between the crooked mutant and the wild type:

```
#define CROOKED 0                                   /* whether I am modeling crooked or wild type */
#define E_JUV_BASE (CROOKED? E_JUV : 3*E_JUV)       /* average Young's modulus for crooked vs. wild type juvenile wood*/
#define E_ADULT_BASE (CROOKED? E_ADULT : 3*E_ADULT) /* crooked vs. wild type adult wood */
#define REACT_ALPHA_BASE (CROOKED? 0.0 : REACT_ALPHA ) /* percentage of reaction wood: none for crooked tree */
```

The amount of tropism is reduced accordingly for the crooked tree.

```
if( !CROOKED ) Tropism *= 5;
```

The function used to calculate the amount of resources granted to a child branch, as described in Equation 7.1, is:

```
/* Heuristic: calculate proportion of parent's resources that child2 gets. */
float VigourProportion( const V3f& hParent, const V3f& hChild, float rParent, float rChild, float yChild, float yStart, float yMax )
{
    float Q = 1.0;
    float favour = 0.0;

    float heightFactor;
    if(yMax == yStart) heightFactor = 0;
    else heightFactor = (yChild - yStart)/(yMax - yStart);
    if( heightFactor < 0 ) heightFactor = 0;
    float orientFactor = DotProd(hChild, up);
    favour = heightFactor * orientFactor;

    float orientation = 2*(1 - fabs(DotProd(hParent, up) )) * DotProd(hChild, up);

    float proportion = pow(rChild, (float)(2.0*PIPE))/pow(rParent, (float)(2.0*PIPE));
    proportion *= favour;

    /* I have found "fair" proportion: now introduce unfairness.
    * This is "winner takes all" way. fraction of fair for smaller proportion, 1-smallerproportion for larger proportion */
    if( proportion > 0.5 )          proportion = ( 1.0 - FAIRNESS*(1-proportion) );
    else                            proportion = FAIRNESS*proportion;

    // Sanity check: clamp at 1 and 0
    if( proportion > 1 )            proportion = 1;
    else if( proportion < 0 )       proportion = 0;

    // Only distribute a certain percentage of resources this year (save some to extend to next year)
    return proportion*Q;
}
```

The production rule to determine whether a bud (`PotentialApex` module) will become a shoot (beginning with conversion to an `Apex` module) is as follows:

```
/* PotentialApex can turn into an apex at next year. Probability of doing
so is based on amount of resources granted.
*/
Internode(idl) << BranchAngle(ph, which, th) Internode(id) PotentialApex(leafed, apexAge) :
{

        id.ageYear++;

        /* Determine amount of resources. Because of order of string interpretation, subtract from a global variable.
         * Set global vigMainBranch if i am the start of the branch */
        if( idl.startOfBuddingBranch )
        {
            vigMainBranch = idl.vigour;
        }

        id.vigour = vigMainBranch * VigourProportion(idl.HLU.H, id.HLU.H, idl.goal_r, id.goal_r, id.Pos.y, id.yStart, idl.yMax );

        /* If the bud shoots, create a sink which removes resources needed to shoot */
        if( BudShoots(id.vigour) )
        {
            float sinkFactor = ChildNumBuds(id.vigour)/4.0;
            vigMainBranch -= id.vigour * sinkFactor;

            /* Sanity check */
            if( vigMainBranch < 0 ) vigMainBranch = 0;

            id.isBud = false;
            id.startOfBuddingBranch = true;
            id.numBudsThisShoot = ChildNumBuds(id.vigour);
            id.yStart = id.Pos.y;
            produce BranchAngle(ph, which, th) Internode(id)
                Apex(0, id.numBudsThisShoot, 0);
        }
        else
        {
            id.takesVigour = false;
            produce BranchAngle(ph, which, th) Internode(id);
        }
}
```

In the `Apex` production rule of a crooked poplar tree, the effect of tropism is reduced over the course of the summer.

```
if(CROOKED)
{
    /* Amount of tropism decreases from start of summer -- so later internodes are less affected */
    float tropFactor = ((float)(3*numBuds - count)/(float)(numBuds));
```

```
    // Clamp at 1
    tropFactor = ( tropFactor > 1 )? 1 : tropFactor;
    Tropism *= tropFactor;
}
```

The values of *yMax* and *yStart* are initialized in the `Apex` production:

```
/* Store the start of this shoot's y position, and also make that the starting yMax, for resources distribution calculation */
id1.yStart = id2.yStart = id.yStart;
/* Straight child should inherit yMax, unless its y value exceeds it */
id1.yMax = Max(id.yMax, id1.Pos.y);
id2.yMax = Max(id.yStart, id2.Pos.y);
```

The maximum height value (*yMax*) for a particular branch is computed in a forward pass:

```
/* straight child only: inherit yMax, unless I can top it */
if( which == 1 ) id.yMax = Max(id1.yMax, id.Pos.y);
else id.yMax = Max(id.yStart, id.Pos.y);

// Send biggest changeInOmega result (which was found in backpass) forward through chain
id.changeInOmega = id1.changeInOmega;

produce BranchAngle(ph, which, th) Internode(id);
```

The computed *yMax* value is propagated along the shoot in a backward pass:

```
/* Pass calculated yMax back along the shoot -- but only from straight child */
id.yMax = id1.yMax;
```

A.3 Architecture control via descriptive functions

This section lists the code changes needed to perform architecture control by using graphical functions modifiable by the user. Functions are called using the convention `func(NAME_OF_FUNCTION)`. Using the function-based strategy, all references to resource allocation and vigour are removed from the model. To determine whether a bud becomes a shoot and how many new internodes this shoot should have, the following procedures are used:

```
/* Determine whether to make a bud into a shoot */
bool BudShoots(int count, int order, float length)
{
    if( func(BRANCH_PROB_COUNT, (float)count/(float)NUMBUDS) * func(BRANCH_PROB_ORDER, (float)order/10.0)
            * func(BRANCH_PROB_LENGTH, length/1.0)  > RAND(0.5, 0.65) )
        return true;
    else
        return false;
}

/* Determine number of internodes to give the child branch. */
float ChildNumBuds(float pLen, int count, int order)
{
    int numBuds = func(LENGTH_COUNT, (float)count/(float)MAXBUDS)
        * func(LENGTH_ORDER, (float)order/10.0)
        * MAXBUDS
        * RAND(0.8, 1.0);

    return numBuds;
}
```

These procedures are called from the `PotentialApex` production rule:

```
/* PotentialApex can turn into an apex at next year. Probability of doing so
* is based on graphical functions set by user.
*/
Internode(idl) << BranchAngle(ph, which, th) Internode(id) PotentialApex(leafed, apexAge) :
{
        id.ageYear++;

        if( BudShoots(id.count, id.order, idl.len) )
        {
            id.isBud = false;
            id.startOfBuddingBranch = true;
            int numBuds = ChildNumBuds(idl.len, id.count, id.order);
            if( numBuds > 1 )
                produce BranchAngle(ph, which, th) Internode(id) Apex(0, ChildNumBuds(idl.len, id.count, id.order), 0);
            else produce BranchAngle(ph, which, th) Internode(id);
        }
        else
        {
            produce BranchAngle(ph, which, th) Internode(id);
        }
}
```

A.4 Dynamics

Given below is the entire LPFG code for the dynamics model described in Chapter 9.

```
#include <lpfgall.h>
#include <math.h>
#include "vectorlib.h"

#define dt 0.005
#define gravity 9.8
#define b 3.2
#define springK 300.0
#define WALL_K 300.0
#define DEG_TO_RAD 0.017453293
#define RAD_TO_DEG 57.29577951

V3d horizontal(1,0,0);
V3d zaxis(0,0,1);
V3d gravityv = V3d(0, -gravity, 0);
V3d v0(0,0,0);

// ----- GROUPS (use scanning direction) -----------------------------
#define FWD     1
#define BKWD        2

struct link {
    V3d alpha;          // orientation
    V3d omega;          // rotational velocity
    V3d epsilon;        // rotational acceleration
    V3d r;              // vector representation of the link's rod
    V3d atv;            // total acceleration of the link (tangential plus radial)
    double m;           // mass
    V3d force;          // force that acts on previous: from radial component of acceleration
    V3d i_force;        // inertial version of above.
    V3d i_forceToLeft;  // force left over to pass to left.
    V3d i_forceFromR;   // accumulated force from the link to the right.
    int which;          // for debugging: which link I am
    double k;           // spring constant for this spring
    V3d rest_angle;     // rest angle for the spring, between left link and me. For now, a vector :-(
    V3d deltaAngleR1;   // branch angle minus rest angle
    V3d deltaAngleR2;
};

/* functions for initializing links */

link first_init()
{
    link s;
    s.alpha = V3d(0,0,0*DEG_TO_RAD);
    s.omega = v0;
    s.epsilon = v0;
    s.r = 1*VecRotateZ(horizontal, s.alpha.z);
    s.force = s.i_force = s.i_forceFromR = s.i_forceToLeft = s.atv = v0;
    s.m = 1.0;
    s.which = 0;
    s.k = springK;
    s.rest_angle = v0;
    return s;
```

```
}
link second_init()
{
    link s;
    s.alpha = V3d(0,0,40*DEG_TO_RAD);
    s.omega = v0;
    s.epsilon = v0;
    s.r = 1*VecRotateZ(horizontal, s.alpha.z);
    s.force = s.i_force = s.i_forceFromR = s.i_forceToLeft = s.atv = v0;
    s.m = 1.0;
    s.which = 0;
    s.k = springK;
    s.rest_angle = V3d(0,0,40*DEG_TO_RAD);
    return s;
}
link third_init()
{
    link s;
    s.alpha = V3d(0,0,-30*DEG_TO_RAD);
    s.omega = v0;
    s.epsilon = v0;
    s.r = 1*VecRotateZ(horizontal, s.alpha.z);
    s.force = s.i_force = s.i_forceFromR = s.i_forceToLeft = s.atv = v0;
    s.m = 1.0;
    s.which = 0;
    s.k = springK;
    s.rest_angle = V3d(0,0,-30*DEG_TO_RAD);
    return s;
}
link fourth_init()
{
    link s;
    s.alpha = V3d(0,0,-60*DEG_TO_RAD);
    s.omega = v0;
    s.epsilon = v0;
    s.r = 1*VecRotateZ(horizontal, s.alpha.z);
    s.force = s.i_force = s.i_forceFromR = s.i_forceToLeft = s.atv = v0;
    s.m = 1.0;
    s.which = 0;
    s.k = springK;
    s.rest_angle = V3d(0,0,-60*DEG_TO_RAD);
    return s;
}
link fifth_init()
{
    link s;
    s.alpha = V3d(0,0,60*DEG_TO_RAD);
    s.omega = v0;
    s.epsilon = v0;
    s.r = 1*VecRotateZ(horizontal, s.alpha.z);
    s.force = s.i_force = s.i_forceFromR = s.i_forceToLeft = s.atv = v0;
    s.m = 1.0;
    s.which = 0;
    s.k = springK;
```

```
    s.rest_angle = V3d(0,0,60*DEG_TO_RAD);
    return s;
}
link tweak_init()
{
    link s;
    s.alpha = V3d(0,0,0*DEG_TO_RAD);
    s.omega = v0;
    s.epsilon = v0;
    s.r = 1*VecRotateZ(horizontal, s.alpha.z);
    s.force = s.i_force = s.i_forceFromR = s.i_forceToLeft = s.atv = v0;
    s.m = 1.0;
    s.which = -1;
    s.k = springK;
    s.rest_angle = V3d(0,0,0*DEG_TO_RAD);
    return s;
}

/* global variables */

double Time;
link first_link_init;
link second_link_init;
link third_link_init;
link fourth_link_init;
link fifth_link_init;
link tweak_link_init;

module P(link);             // straight link with a mass and a rotational spring at its distal end.
module BA(V3d);             // branching angle

int dir;

Start: {Time=0;
    first_link_init = first_init();
    second_link_init = second_init();
    third_link_init = third_init();
    fourth_link_init = fourth_init();
    fifth_link_init = fifth_init();
    tweak_link_init = tweak_init();
    dir=BKWD;
    Backward();
    UseGroup(dir);
}

StartEach: {Time+=dt;
    UseGroup(dir);
    if (dir==FWD)
        Forward();
    if (dir==BKWD)
        Backward();
    dir = 3-dir;
}
```

```
derivation length: 0;

Axiom: Right(90) P(first_link_init) BA(v0) P(first_link_init) BA(v0) P(first_link_init)
            // Specify the branched structure
            SB() BA(v0) P(second_link_init)
                    SB() BA(v0) P(fifth_link_init) BA(v0) P(first_link_init) BA(v0) P(first_link_init) EB()
                    SB() BA(v0) P(first_link_init) EB()
            EB()
            SB() BA(v0) P(first_link_init)
                SB() BA(v0) P(first_link_init)
                    SB() BA(v0) P(first_link_init) EB()
                    SB() BA(v0) P(third_link_init) EB()
                EB()
                SB() BA(v0) P(fifth_link_init) BA(v0) P(tweak_link_init) EB()
            EB();

group BKWD: //-------------------------------------------------------------------------------


/* Two children, a branching point */
P(s) >> SB()  BA(branch_angle1) P(sr1) /*...*/ EB()
    SB()  BA(branch_angle2) P(sr2) /*...*/ :
{
    /* Accumulate the unused forces from the right, not including my own force. */
    s.i_forceFromR = sr1.i_forceToLeft + sr2.i_forceToLeft;

    /* Save the (branch angle - rest angle) to the right, to compute spring torque from R */
    s.deltaAngleR1 = (branch_angle1 - sr1.rest_angle);
    s.deltaAngleR2 = (branch_angle2 - sr2.rest_angle);

    produce P(s);
}

/* One child */
P(s) >> BA(branch_angle) P(sr) :
{
    /* accumulate the unused forces from the right, not including my own force. */
    s.i_forceFromR = sr.i_forceToLeft;

    /* Save the (branch angle - rest angle) to the right, for spring torque from R */
    s.deltaAngleR1 = (branch_angle - sr.rest_angle);
    s.deltaAngleR2 = v0;    //There is no second child

    produce P(s);
}

/*  Rightmost link */
P(s) :
{
    s.i_forceFromR = v0; //no right neighbour
    if( s.which == -1 && ((Time >= 13 && Time < 13.05)
                || (Time >= 20 && Time < 20.05)
                || (Time >= 30 && Time < 30.05))
    )
```

```
        s.i_forceFromR = V3d(0, -400, 0);

    /* Save the (branch angle - rest angle) to the right, for spring torque from R */
    s.deltaAngleR1 = s.deltaAngleR2 = v0;

    produce P(s);
}

group FWD: //--------------------------------------------------------------------------------

/* Branching angle is updated at every step. For easier future integration with static model */
P(sl) << BA(branch_angle) > P(sr) :
{
    branch_angle = sr.alpha - sl.alpha;
    produce BA(branch_angle);
}

P(sl) BA(branch_angle) << P(s) :
{
    /* Spring torques, based on relative angles between successive segments.
    * Torques from R will be zero (since deltaAngle will be zero) if there are no corresponding right children. */
    V3d torqueFromLeftSpring = -sl.k * (branch_angle - s.rest_angle);
    V3d torqueFromRightSpring1 = s.k * s.deltaAngleR1;
    V3d torqueFromRightSpring2 = s.k * s.deltaAngleR2;
    V3d springTorque = torqueFromLeftSpring + torqueFromRightSpring1 + torqueFromRightSpring2;

    double I = s.m*s.r.Length()*s.r.Length();


    /* To get total force stretching the segment, take parallel components with respect to s.r of
    * contributing accelerations. */
    V3d ni_total_force = -1*s.m*sl.atv + s.m*gravityv + s.i_forceFromR + s.m*((springTorque/I) % s.r)  - (b*s.omega % s.r) ;

    s.epsilon = ((s.r % (ni_total_force/s.m)) / (s.r.Length()*s.r.Length()));
    s.omega += s.epsilon*dt;
    s.alpha += s.omega*dt;
    s.r = s.r.Length()*VecRotate(horizontal, zaxis, s.alpha.z);

    /* Make the inertial version of the force: using parallel component of RF's inertial acceleration */
    s.i_force = s.m*sl.atv + ni_total_force;

    /* Centripetal acceleration of distal point, and
    * tangential acceleration which has been reduced by damping
    */
    V3d av = s.omega % (s.omega % s.r);
    V3d tv = (s.epsilon % s.r);

    /* This is the inertial acceleration (accumulated from LHS) */
    s.atv = av + tv + sl.atv;

    /* Force left over to pass to left */
    s.i_forceToLeft = ParallelComponent(s.i_force - s.m*s.atv, s.r);

    produce P(s);
```

```
}

// Leftmost link
P(s) :
{
    /* Spring torques, based on relative angles between successive segments.
    * Torques from R will be zero (since deltaAngle will be zero) if there are no corresponding right children. */
    V3d torqueFromLeftSpring = -1* WALL_K * (s.alpha);
    V3d torqueFromRightSpring1 = s.k * s.deltaAngleR1;
    V3d torqueFromRightSpring2 = s.k * s.deltaAngleR2;
    V3d springTorque = torqueFromLeftSpring + torqueFromRightSpring1 + torqueFromRightSpring2;

    double I = s.m*s.r.Length()*s.r.Length();

    /* To get total force stretching the segment, take parallel components with respect to s.r of
    * contributing accelerations.
    */
    V3d ni_total_force = s.m*gravityv + s.i_forceFromR + s.m*((springTorque/I) % s.r) - (b*s.omega % s.r) ;

    s.epsilon = ((s.r % (ni_total_force/s.m)) / (s.r.Length()*s.r.Length()));
    s.omega += s.epsilon*dt;
    s.alpha += s.omega*dt;
    s.r = s.r.Length()*VecRotate(horizontal, zaxis, s.alpha.z);

    s.i_force = ni_total_force;


    /* Centripetal acceleration of distal point, and
    * tangential acceleration which has been reduced by damping
    */
    V3d av = s.omega % (s.omega % s.r);
    V3d tv = (s.epsilon % s.r);

    /* Leftmost link has no parent, so this is inertial acceleration */
    s.atv = av + tv;

    /* Force left over to pass to left */
    s.i_forceToLeft = ParallelComponent(s.i_force - s.m*s.atv, s.r);

    produce P(s);
}


group 0:  //-------------------------------------------------------------------------------

interpretation:

P(l) :
{
    produce
        SetColor(1) SetWidth(0.03) LineRel3d(l.r) SetColor(4) Sphere(0.1);
}
```

Bibliography

[1] L D Landau L P Pitaevskii A M Kosevich, E M Lifshitz. *Theory of Elasticity : Volume 7 (Theoretical Physics, Vol 7)*. Reed Elsevier, 1986.

[2] Mitch Allen, Przemyslaw Prusinkiewicz, and Theodore DeJong. Using L-systems for modelling source-sink interactions, architecture and physiology of growing trees: the L-PEACH model. *New Phytologist*, 166:869 – 880, 2005.

[3] Tancréde Alméras, Joseph Gril, and Evelyne Costes. Bending of apricot tree branches under the weight of axillary growth: test of a mechanical model with experimental data. *Trees: Structure and Function*, 16(3):5–15, 2002.

[4] Ken-ichi Anjyo, Yoshiaki Usami, and Tsuneya Kurihara. A simple method for extracting the natural beauty of hair. *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 111–120, 1992.

[5] R. Archer. *Springer Series in Wood Science*, chapter Growth stresses and strains in trees. Springer, 1986.

[6] Vernon Barger and Martin Olsson. *Classical Mechanics: A Modern Perspective*. McGraw-Hill, 1995.

[7] Heike Beismann, Hiltrud Wilhelmi, Henri Baillères, Hanns-Christof Spatz, Arno Bogenrieder, and Thomas Speck. Brittleness of twig bases in the genus *Salix*: fracture mechanics and ecological relevance. *Journal of Experimental Botany*, 51(344):617 – 633, March 2000.

[8] Rolf Borchert and Hisao Honda. Control of development in the bifurcating branch system of tabebuia rosea: a computer simulation. *Botanical Gazette*, 145(2):184–195, 1984.

[9] P. C. Chou and N.J. Pagano. *Elasticity: Tensor, Dyadic, and Engineering Approaches*. Dover, New York, 1992.

[10] J. Craig. *Introduction to Robotics, Mechanics and Control*. Addison-Wesley, 1989.

[11] Agnes Daldegan, Nadia Magnenat-Thalmann, Tsuneya Kurihara, and Daniel Thalmann. An integrated system for modelling, animating and rendering hair. *Computer Graphics Forum*, 12(3):211–221, 1993.

[12] Oliver Deussen, Bernd Lintermann, and A. Dowden-Williams. *Digital Design of Nature : Computer Generated Plants and Organics*. Springer, Berlin, 2005.

[13] T. Fourcaud and P. Lac. Numerical modelling of shape regulation and growth stresses in trees. *Trees - Structure and Function*, 17:23–30, 2002.

[14] Meriem Fournier. Mécanique de l'arbre. maturation, poids propre, contraintes climatiques. *Thèse de Doctorat, Institut National Polytechnique de Lorraine*, 1989.

[15] Meriem Fournier, Henri Bailleres, and Bernard Chanson. Tree biomechanics: Growth, cumulative prestresses, and reorientations. *Biomimetics*, 2(3):229–251, 1994.

[16] D. W. Green. Wood: strength and stiffness. *Encyclopedia of Materials: Science and Technology*, pages 9732 – 9736, 2001.

[17] Henri D. Grissino-Mayer, 2004. http://web.utk.edu/ grissino/gallery.htm.

[18] Sunil Hadap and Nadia Magnenat-Thalmann. Modeling dynamic hair as a continuum. *Eurographics*, 20(3), 2001.

[19] David Halliday and Robert Resnick. *Fundamentals of Physics*. John Wiley and Sons, Inc, 2nd edition edition, 1981.

[20] John Hart and Brent Baker. Implicit modeling of tree surfaces. *Proceedings of the 1996 workshop on implicit surfaces*, 1996.

[21] John Hart, Brent Baker, and Jeyprakash Michaelraj. Structural simulation of tree growth and response. *The Visual Computer*, 19:151–163, 2003.

[22] R.C. Hibbeler. *Statics and Mechanics of Materials*. Macmillan Publishing, 1992.

[23] Matthew Holton. Strands, gravity and botanical tree imagery. *Computer Graphics Forum*, 13(1), 1994.

[24] H. Honda, P. B. Tomlinson, and J. B. Fisher. Computer simulation of branch interaction and regulation by uneven flow rates in botanical trees. *American Journal of Botany*, 68(4):569–585, 1981.

[25] Catherine Alena Jirasek. A biomechanical model of branch shape in plants expressed using L-systems. *Master's Thesis, University of Calgary*, 2000.

[26] Huw Jones and Matthew Aitken. Modelling the effect of wind blown trees. *Eurographics UK Chapter Annual Conference*, pages 245–254, 1997.

[27] Wilfred A. Côté Jr. *Wood Ultrastructure: An Atlas of Electron Micrographs*. University of Washington Press, 1967.

[28] Radoslaw Karwowski. Improving the process of plant modelling: the L+C modelling language. *Ph.D. Dissertation, University of Calgary*, 2002.

[29] Radoslaw Karwowski and Przemyslaw Prusinkiewicz. Design and implementation of the l+c modelling language. *Electronic Notes in Theoretical Computer Science*, 86(2), 2003.

[30] C. H. A. Little. Some aspects of apical dominance in *Pinus Strobus*. *Ph.D. Dissertation*, 1967.

[31] Richard E. Mark. *Cell Wall Mechanics of Tracheids*. Yale University Press, 1967.

[32] C. Mattheck and H. Kubler. *Wood: The Internal Optimization of Trees*. Springer-Verlag, 1997.

[33] Claus Mattheck. *Design in Nature: Learning from Trees*. Springer-Verlag, 1998.

[34] C. Murray. A relationship between circumference and weight in trees and its bearing on branching angles. *Journal of General Physiology*, 10:725–729, 1927.

[35] Karl J. Niklas. *Plant Biomechanics: An Engineering Approach to Plant Form and Function*. University of Chicago Press, 1992.

[36] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[37] Przemyslaw Prusinkiewicz, August, 2005. Personal communication.

[38] Przemyslaw Prusinkiewicz, Mark Hammel, Jim Hanan, and Radomír Mech. *Handbook of Formal Languages*, chapter Visual Models of Plant Development. Springer-Verlag, 1996.

[39] Przemyslaw Prusinkiewicz, Catherine Alena Jirasek, and B.Moulia. Integrating biomechanics into developmental plant models expressed using L-systems. *Plant biomechanics*, 191:615–624, 1998.

[40] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer, 1996.

[41] Przemyslaw Prusinkiewicz, Lars Mündermann, Radoslaw Karwowski, and Brendan Lane. The use of positional information in the modelling of plants. *Proceedings of SIGGRAPH*, pages 289–300, 2001.

[42] William Remphrey, May, 2005. Personal communication.

[43] William R. Remphrey and Linda P. Pearn. Crown development of a clone of populus tremuloides exhibiting "crooked" architecture and a comparison with wild-type trees. *Canadian Journal of Botany*, 81:345–359, 2003.

[44] J. P. Richter, editor. *The Literary Works of Leonardo Da Vinci*. Phaidon Publishers Inc., 1970.

[45] Robert Rosenblum, Wayne Carlson, and Edwin Tripp. Simulating the structure and dynamics of human hair: Modeling, rendering and animation. *Journal of Visualzation and Computer Animation*, 2:141Ű148, 1991.

[46] T. Sakaguchi and J. Ohya. Modelling and animation of botanical trees for interactive virtual environments. *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 139–146, 1999.

[47] K. Shinozaki, K. Yoda, K. Hozumi, and T. Kira. A quantitative analysis of plant form - the pipe theory model. i. basic analyses. *Japanese Journal of Ecology*, 14(3):97–105, 1964.

[48] M. Shinya and A. Fournier. Stochastic motion - motion under the influence of wind. *Proceedings of EUROGRAPHICS 1992*, 11:C119–C128, 1992.

[49] Wendy Kuhn Silk. Kinematics and dynamics of primary growth. *Biomimetics*, 2(3):199–213, 1994.

[50] J. Stam. Stochastic dynamics: Simulating the effects of turbulence on flexible structures. *Computer Graphics Forum*, 16:159–164, August 1997.

[51] Julia Taylor-Hell. Incorporating biomechanics into architectural tree models. *Proceedings of the 18th Brazilian Symposium on Computer Graphics and Image Processing*,

2005.

[52] Eric W. Weisstein, July, 2005. http://scienceworld.wolfram.com/physics/Pendulum.html.

[53] Brayton F. Wilson. *The Growing Tree*. University of Massachusetts Press, 1984.

[54] Brayton F. Wilson. Apical control of branch growth and angle in woody plants. *American Journal of Botany*, 87:601–607, 2000.