Abstract

This thesis presents an attempt to model extinct plants described by fossil data. The well-preserved plants of the Rhynie chert in northeast Scotland were chosen due to the amount of information available and their scientific significance. These plants are modelled with L-system techniques developed to model extant plant species.

Fossil specimens of each plant species and the observations of palaeobotanists working with these fossils were used to reconstruct different stages of development of the Rhynie chert plants, and model their growth. These represent the first dynamic models of these ancient plants, since previous models have been static drawings or sculptures.

Ecological information about the Rhynie chert plants is also available. This information was used to create an ecosystem model that simulates the interactions between the plants and the environment. The ecosystem model and the individual plant models were then combined into final images that represent the overall appearance of the ancient Rhynie chert.
Acknowledgements

During my Master’s degree at the University of Calgary, countless individuals have provided me with academic and personal support. Without them, I would not have made it through the gruelling writing process. I would like to extend my thanks to all of them.

First, I would like to thank my supervisor, Dr. Przemyslaw Prusinkiewicz for his support and constructive criticisms, as well as his patience with my perfectionism. I would also like to thank the rest of the lab, especially Adam Runions and Mik Cieslak, for their academic and moral support, and Elizabeth Barker for her helpful advice and limitless supply of useful papers.

Secondly, I’d like to thank my friend and almost Master’s degree holder, Scott Beach, for ensuring that I had the courage to see my degree through to completion.

Finally, I have to thank my family for putting up with my student shenanigans for far longer than they ever expected to. They provided both academic and moral support, and provided a much needed supply of chocolate in dire situations.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>iii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>iv</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>1. Introduction</td>
<td></td>
</tr>
<tr>
<td>1.1 Objectives</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Organization</td>
<td>3</td>
</tr>
<tr>
<td>2. Background</td>
<td></td>
</tr>
<tr>
<td>2.1 The Origin of Plants</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 The Evolution of Plants</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2 Colonization of Land</td>
<td>5</td>
</tr>
<tr>
<td>2.1.3 Evolution of Vasculature</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Geologic Setting</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Significance to Science</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Previous Modelling Work</td>
<td></td>
</tr>
<tr>
<td>2.4.1 Plant Models</td>
<td>9</td>
</tr>
<tr>
<td>2.4.2 Ecosystem Models</td>
<td>12</td>
</tr>
<tr>
<td>3. Modelling Methods</td>
<td></td>
</tr>
<tr>
<td>3.1 L-Systems</td>
<td>15</td>
</tr>
<tr>
<td>3.1.1 Parametric L-Systems</td>
<td>17</td>
</tr>
<tr>
<td>3.1.2 Context Sensitive L-Systems</td>
<td>18</td>
</tr>
<tr>
<td>3.1.3 Environmentally Sensitive L-Systems</td>
<td>18</td>
</tr>
<tr>
<td>3.2 L-studio</td>
<td>20</td>
</tr>
<tr>
<td>3.2.1 cpfg and lpfg</td>
<td>20</td>
</tr>
<tr>
<td>3.2.2 Functions, Contours and Surfaces</td>
<td>23</td>
</tr>
<tr>
<td>3.2.3 Tropism and Elasticity</td>
<td>25</td>
</tr>
<tr>
<td>3.3 Terminology</td>
<td>25</td>
</tr>
<tr>
<td>4. Plants of the Rhynie Chert</td>
<td></td>
</tr>
<tr>
<td>4.1 General Information</td>
<td>29</td>
</tr>
<tr>
<td>4.2 General Modelling Techniques</td>
<td>29</td>
</tr>
<tr>
<td>4.3 <em>Horneophyton lignieri</em></td>
<td></td>
</tr>
<tr>
<td>4.3.1 Description</td>
<td>33</td>
</tr>
<tr>
<td>4.3.2 Modelling</td>
<td>34</td>
</tr>
<tr>
<td>4.3.3 Implementation</td>
<td>38</td>
</tr>
<tr>
<td>4.3.4 Results</td>
<td>46</td>
</tr>
<tr>
<td>4.4 <em>Aglaophyton major</em></td>
<td></td>
</tr>
<tr>
<td>4.4.1 Description</td>
<td>48</td>
</tr>
<tr>
<td>4.4.2 Modelling</td>
<td>52</td>
</tr>
<tr>
<td>4.4.3 Implementation</td>
<td>53</td>
</tr>
<tr>
<td>4.4.4 Results</td>
<td>59</td>
</tr>
<tr>
<td>4.5 <em>Rhynia gwynne-vaughanii</em></td>
<td>62</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Important symbols and modules in cpfg and lpfg</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Selected previous reconstructions of <em>Horneophyton</em>, and some of their drawbacks</td>
<td>35</td>
</tr>
<tr>
<td>4.2</td>
<td>Values and sources of important modelling parameters for <em>Horneophyton</em></td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>A list of the L-System modules used to model <em>Horneophyton</em></td>
<td>40</td>
</tr>
<tr>
<td>4.4</td>
<td>Functions used to model different aspects of <em>Horneophyton</em>: $\text{growthRate}(age)$ returns the amount that a section of the plant will increase in length, which slows over time. $\text{colour}(age)$ returns the index to a colour in the colourmap. $\text{vigourDecay}(yh)$ returns a factor by which the current vigour will be multiplied, which is dependent on the orientation of the apex. $\text{rhizomeWidth}(n)$ returns the width of 'rhizome' segment $n$. See Fig. 4.7 for base functions, and 4.8 for definition of $C_1$ and $C_n$, $a$, $b$, $c$, and $factor$ are constants</td>
<td>42</td>
</tr>
<tr>
<td>4.5</td>
<td>Values and sources of important modelling parameters for <em>Aglaophyton</em></td>
<td>52</td>
</tr>
<tr>
<td>4.6</td>
<td>A list of the L-System modules used to model <em>Aglaophyton</em></td>
<td>54</td>
</tr>
<tr>
<td>4.7</td>
<td>Functions used to model different aspects of <em>Aglaophyton</em>: $\text{sporangiumWidth}(n)$ returns the width of the sporangium at each segment $n$, and $\text{sporangiumSplit}(n,age)$ returns the angle between each segment $n$, which changes over time. See Fig. 4.22 for base functions</td>
<td>54</td>
</tr>
<tr>
<td>4.8</td>
<td>Values and sources of important modelling parameters for <em>Rhynia</em></td>
<td>64</td>
</tr>
<tr>
<td>4.9</td>
<td>A list of the L-System modules used to model <em>Rhynia</em></td>
<td>65</td>
</tr>
<tr>
<td>4.10</td>
<td>Values and sources of important modelling parameters for <em>Nothia</em></td>
<td>73</td>
</tr>
<tr>
<td>4.11</td>
<td>A list of the L-System modules used to model <em>Nothia</em></td>
<td>75</td>
</tr>
<tr>
<td>4.12</td>
<td>Values and sources of important modelling parameters for <em>Ventarura</em></td>
<td>82</td>
</tr>
<tr>
<td>4.13</td>
<td>A list of the L-System modules used to model <em>Ventarura</em></td>
<td>83</td>
</tr>
<tr>
<td>4.14</td>
<td>Values and sources of important modelling parameters for <em>Trichopherophyton</em></td>
<td>89</td>
</tr>
<tr>
<td>4.15</td>
<td>A list of the L-System modules used to model <em>Trichopherophyton</em></td>
<td>90</td>
</tr>
<tr>
<td>4.16</td>
<td>A list of the L-System modules used to model <em>Asteroxylon</em></td>
<td>98</td>
</tr>
<tr>
<td>4.17</td>
<td>Functions used to model different aspects of <em>Asteroxylon</em>: $a$, $factor$, and $maxOrder$ are constants. See Fig. 4.57 for base functions</td>
<td>98</td>
</tr>
<tr>
<td>4.18</td>
<td>Values and sources of important modelling parameters for <em>Asteroxylon</em></td>
<td>99</td>
</tr>
<tr>
<td>5.1</td>
<td>The ideal ranges for each simulation parameter for each plant. Minimum and maximum values can be interpreted as the minimum required conditions and the maximum tolerated conditions. If all parameters are inside the ideal range, the plant is healthy. The plant suffers if any parameter is outside the ideal range, and will fail to germinate on substrates that are too hostile</td>
<td>104</td>
</tr>
</tbody>
</table>
List of Figures and Illustrations

2.1 A. Estimated palaeogeographic location of the Rhynie Chert. Image modified after [7]. B., C. Modern day location of the Rhynie Chert. Images courtesy of NASA. ................................................................. 8

3.1 An example of a basic L-system which produces a simple tree-like branching structure. ................................................................. 16

3.2 The quadratic Koch curve at iterations 1, 2 and 4. This fractal can be generated by an L-system with only one production rule, $F \rightarrow F+F-F-FF+F+F-F$. .............................................. 16

3.3 An example of a parametric L-system, which produces a tree-like branching structure with organs and internodes of varying size. .......................... 17

3.4 An example of a context sensitive, stochastic L-system, which produces a tree-like branching structure with a more organic feel. Both sympodial and monopodial branching is possible via two productions for $A$. They will be selected with probabilities $\frac{3}{4}$ and $\frac{1}{4}$, respectively. The productions for $B$ will be selected based on whether or not $A(y)$ or $[B(xR, wR)]B(xL, wL)]$ exists in its right context. In the latter case, $B$ modules preceding a branching point will increase in width according to the pipe model. ........................................... 19

3.5 A tree-like structure generated by running eleven iterations of the simple L-system in Fig. 3.4, after replacing the $180^\circ$ roll with a $137.5^\circ$ roll to make the result three-dimensional. .................................................. 20

3.6 An example of an environmentally sensitive L-system. The colours of the background represent the availability of different soil nutrients, while the coloured spheres represent various species of plant which grow more rapidly under different conditions. .................................................. 21

3.7 An example of an L-system written in *lpfg*. This L-system is equivalent to the *cpfg*-based example shown in Fig. 3.4. The consider statements specify which modules are noticed when using context-sensitive productions. .......... 22

3.8 An example of a function, a contour, and a surface in L-studio. .................. 24

3.9 Three axes with the same tropism direction and different elasticity values. .................. 24

4.1 A diagram of a generic Rhynie plant. The red lines indicate the location of the ground depending on whether the plant’s horizontal axes are subterranean or subaerial. Sections of the plant are labelled as follows: $A$ shoot apices, $I$ aerial internodes, $S$ possible sporangium-forming region (lateral or terminal), $A_R$ rhizome apex, $I_R$ rhizome internode, $B$ plant base (spore location), $t_t$ location of termination threshold in terms of apex vigour, $t_b$ location of branching threshold in terms of apex vigour. ................................. 30

4.2 Transverse sections of *Asterozylon* showing the phyllotactic arrangement of its enations and internal vasculature. Images taken from [3]. .................. 31

4.3 A. Death of *Horneophyton* by shrivelling. The existing bends within the axes are exaggerated over time. B. Death of *Horneophyton* by withering. All axes bend downwards. .................................................. 33
4.4 A fossil rhizomatous axis of *Horneophyton*, compared to B, its counterpart in the model. The red line indicates the assumed ground level. A. Modified after images from [81]. .............................................. 34

4.5 A. Comparison of a cross-section of a three-lobed *Horneophyton* sporangium to its counterpart in the model. B. Comparison of a cross-section of a two-lobed *Horneophyton* sporangium to its counterpart in the model. Red lines indicate dichotomies in the sporangia. Plates (left) modified from [26]. Images from models (right) edited to remove artefacts. ...................................................... 36

4.6 The contour lines used to give the rhizomatous axes of *Horneophyton* a more realistic shape. ............................................................... 38

4.7 The base functions used to define different aspects of *Horneophyton*’s form. A. *growthRateF*, used to smoothly grow each apex and internode. B. *colourF*, used to smoothly interpolate between different colours in the model. C. *vigour-DecayF*, a probability distribution used to determine the percentage of an apex’s vigour that will remain after each time step. D. *rhizomeWidthF*, an approximation of the longitudinal section of *Horneophyton*’s rhizomatous axes. 39

4.8 Possible colours of *Horneophyton*, selected by the *colourF* function. 40

4.9 The contour lines used to give the sporangia of *Horneophyton* their irregular surface. ................................................................. 40

4.10 A diagram showing the structure of *Horneophyton*’s rhizomatous axes, including a simplified L-system interpretation. Sections of the rhizome are labelled as follows: *Ir*(w,b) rhizome interior segments, with width factor and base width. *Ar*(s) rhizome ‘apex’, with number of segments. A aerial apex. *I*(w) aerial internode, with width. *f*(s) represents the ‘rhizomeWidth’ function (Table 4.4). .................................................. 41

4.11 A diagram showing the structure of *Horneophyton*’s aerial axes, including a simplified L-system interpretation. Sections of the axis are labelled as follows: *I*(w) internode segments, with width. +, -(θ) bends by angle θ. \( \theta \) roll by angle θ. A apices, with vigour and number of segments. S sporangium. *t*<sub>a</sub>, *t*<sub>b</sub> are the termination and branching thresholds, and *x* is the threshold number of internode segments before branching. θ<sub>B</sub> is the plant’s branching angle. θ<sub>D</sub> is the deflection angle. θ<sub>P</sub> is the phyllotactic divergence angle. *f* represents the ‘vigour-Decay’ function. ............................................. 43

4.12 A reconstruction of the bag-like form of a *Horneophyton* sporangium [87]. 44

4.13 A single aerial axis of *Horneophyton* as it appears in the model. 46

4.14 *Horneophyton* at different stages during its development. 47

4.15 A closer view of some of *Horneophyton*’s odd shaped sporangia, modelled within the limitations of L-systems. Right image edited to remove artefacts. 47

4.16 A.B. Fossil axes of *Aglaophyton*, showing different angles of dichotomy. C.D. Details of *Aglaophyton* axes as they appear in the model. Modified after plates from [24]. ............................................................ 48

4.17 A. Fossil decumbent axis of *Aglaophyton*. B. A similar decumbent axis in the model C. Fossil decumbent axis of *Aglaophyton*, showing an upturned aerial axis in anatomical connection. D. A similar axis in the model. Modified after plates from [24]. ............................................................ 49
4.18 A. Longitudinal section of an *Aglaophyton* sporangium, terminally connected to an axis. B. A similar *Aglaophyton* sporangium as it appears in the model. C. A shrivelled, detached sporangium. D. A similar *Aglaophyton* sporangium as it appears in the model. Modified after plates from [24].

4.19 Selected previous reconstructions of *Aglaophyton*. Images from [24], [78] and [6] (with permission) respectively.

4.20 A diagram showing the structure of *Aglaophyton*’s decumbent axes, including a simplified L-system interpretation. Sections of the decumbent axis are labelled as follows: Ir decumbent axis interior segments. Ar(cb) decumbent axis apex, with ‘canBranch’ parameter. A aerial apex. I aerial internode. Red, orange and yellow apices signify ‘growing’, ‘reacting’ and ‘rebounding’ growth states respectively. The circles in the internodes show what the state was at that point, while a slash signifies that ‘canBranch’ is false. $\theta_A$ is the angle between decumbent and aerial axes. $\theta_G$ is the angle the axis bends downward during growth. $\theta_R$ is the angle the axis bends upward when reacting. $\theta_B$ is the angle the axis bends upward when rebounding. $\theta_t$ is the angle between the apex and the ground, and $t$ is a threshold of $\sim 45^\circ$. The dotted black line indicates ground level.

4.21 A diagram showing the structure of *Aglaophyton*’s sporangia, including a simplified L-system interpretation. Sections of the structure are labelled as follows: I aerial internode. Sg sporangium group. S sporangium placeholder. Sh half sporangium. Ss sporangium stalk. $\pm,(-\theta)$ bends by angle $\theta$. \(\theta\) roll by angle $\theta$. $\theta_B$ is the branching angle between sporangial stalks. $P_1$ and $P_2$ are the probabilities of forming a pair of sporangia or a single sporangium, respectively. $f$ is a function dictating the bend between each half sporangium segment over time. Ss $\rightarrow$ F F F is used because a single segment cannot exhibit orthotropism.

4.22 The base functions used to define different aspects of *Aglaophyton*’s form. A. *sporangium*WidthF, an approximation of the longitudinal section of *Aglaophyton*’s sporangia. B. *sporangium*SplitF, causes the sporangia to rapidly but smoothly split open after some time.

4.23 A single model of an *Aglaophyton* plant with multiple aerial axes arising from a system of decumbent axes.

4.24 Details of *Aglaophyton*’s decumbent axes and dehiscent sporangia.

4.25 *Aglaophyton* at different stages during its development.

4.26 Reconstructed fossil axes of *Rhynia*, compared to similar axes from the model. Modified after plates from [22].

4.27 Selected previous reconstructions of *Rhynia*. Images from [22], [4] and [31] respectively.
4.28 A diagram showing the structure of Rhynia’s decumbent axes, including a simplified L-system interpretation. Sections of the decumbent axis are labelled as follows: Ir decumbent axis interior segments. Ar(1b) decumbent axis apex, with ‘lastBranch’ parameter. A aerial apex. I aerial internode. \( \theta \) is the angle between decumbent and aerial axes. \( \theta_G \) is the angle the axis bends downward during growth. \( \theta_R \) is the angle the axis bends upward when reacting. \( t \) is a threshold of that determines when an aerial axis will be produced. The dotted black line indicates ground level.

4.29 Rhynia at different stages during its development.

4.30 A single model of a Rhynia plant with multiple aerial axes arising from a system of decumbent axes.

4.31 A group of aerial axes of Rhynia bearing sporangia.

4.32 A fossilized aerial axis of Nothia, compared to its counterpart in the model. Fossil image taken from [39].

4.33 Previous reconstructions of Nothia sporangial arrangements (left), modified after diagrams from [27], compared with the model (right). A. B. Columns of sporangia. C. Sporangia arranged in a whorl. D. A terminal cluster.

4.34 A fossilized sporangium of Nothia, showing lateral dehiscence, modified after plates from [27]. B. A sporangium from the model.

4.35 Selected previous reconstructions of Nothia. Images from [27], [39] and [82] respectively.

4.36 The contour lines used to give the sporangia and axes of Nothia their shape.

4.37 The base functions used to define different aspects of Nothia’s form. A. sporangiumWidthF, an approximation of the longitudinal section of Nothia’s sporangium. B. sporangiumSplitF, causes the sporangia to rapidly but smoothly split open after some time.

4.38 A group of aerial axes of Nothia, as they appear in the model. The group is connected together by subterranean rhizomatous axes.

4.39 A close-up of Nothia’s irregular surface and sporangia.

4.40 Nothia at different stages during its development.

4.41 A. A column of fossil Ventarura sporangia, from [57]. B. A single sporangium fossil, showing the line of dehiscence, from [85]. C. A column of sporangia from the model for comparison.

4.42 Preliminary reconstruction of Ventarura, modified after [85].

4.43 The width function and cylinder contour used to give Ventarura’s sporangia their shape.

4.44 Ventarura at different stages during its development.

4.45 A single aerial axis of Ventarura as it appears in the model.

4.46 A close-up of an aerial axis of Ventarura, bearing dehisced sporangia.

4.47 A. Longitudinal section showing the spinose projections of Trichopherophyton [84]. B. An axis with spines as it appears in the model.

4.48 A. Longitudinal section showing a circinate apex of Trichopherophyton [49]. B. A partially coiled apex as it appears in the model.
The base functions used to define different aspects of *Trichopherophyton*’s form. 

A. *sporangiumWidthF*, an approximation of the longitudinal section of *Trichopherophyton*’s sporangia. 

B. *sporangiumSplitF*, causes the sporangia to rapidly but smoothly split open after some time. 

C. *vernationF*, dictates the angle at which each segment of the vernation curls, increasing over its length to create a tight spiral. 

D. *unfurlF*, controls the expression of the vernation angle over time.

Contour representing the transverse section of one valve of a *Trichopherophyton* sporangium.

*Trichopherophyton* at different stages during its development.

A mature aerial axis of *Trichopherophyton* as it appears in the model.

A close-up of partially unfurled branches of *Trichopherophyton*.

Sporangia in the model compared with the only fossil evidence of sporangia in anatomical connection with the main axis. Modified after an image from [48].

Fossil showing the pattern of *Asteroxylon*’s enations laterally and transversely, compared with a top view of the model. Modified after plates from [48] and [49].

Selected previous reconstructions of *Asteroxylon*. Details are regularly filled in using extant clubmosses as a guide. Images from [43], [79] and [34] (with permission) respectively.

The base functions used to define different aspects of *Asteroxylon*’s form.

A. *ageBendF*, defines the angle of each enation over time, causing them to ‘unfold’ from the apex. 

B. *ageFactor*, affects the width of the internodes over time. 

C. *enationBendF*, together with *ageBendF*, defines the angle of the enations. 

D. *enationGrowthF*, defines the size of the enations over time. 

E. *orderReductionF*, used in other functions to reduce the size of enations and axes of higher order branches. 

F. *widthF*, an approximation of the longitudinal section of an axis, defining the width over the length of modelled axes.

A model of a single aerial axis of *Asteroxylon*.

A close-up of the main axis of *Asteroxylon*.

*Asteroxylon* at different stages during its development.

A. The texture-mapped terrain viewed from above, with colours representing the actual appearance of the substrate. The area shown is approximately five metres square. This terrain was generated using EarthSculptor [75]. 

B. An image representing the availability of resources across a section of the above terrain. Each colour represents the availability of one resource from 0 to 1. Water is represented by blue, litter by green, and sinter by red. The black regions represent areas where spores cannot germinate (a moving stream and a hot geothermal pool). Fig. 5.2 shows a visual representation of the each plant’s preference for these colours.

Each plant’s preferred parameter range represented by the colours in Fig. 5.1. Plants grow and spread more successfully on substrates of a similar colour in the simulation.
5.3 An example of a simplified ecosystem after 5, 100, 500 and 1500 iterations. Each of the seven species are present, represented by coloured circles. Species often fail to compete under certain local environmental parameters, and form sustainable populations in more suitable regions. The terrain, and its colour representation, are identical to that shown in Fig. 5.1.

5.4 A close-up of a dense stand of Rhynie chert plants, predominantly (1) *Aglao-phyton* and (2) *Asteroxylon*.


5.6 A different angle on a denser plant community, comprising predominantly (1) *Aglaophyton*, (2) *Asteroxylon*, (3) *Nothia*, (4) *Rhynia*, and (5) *Trichopherophyton*. 
Chapter 1

Introduction

Terrestrial plant life first appeared on Earth 470 million years ago, during the mid-Ordovician period, and has been constantly changing ever since. Gaining a better understanding of the morphology, development and evolution of extinct terrestrial plants can spark insights as to why later prehistoric plants and modern plants exhibit certain structures and traits. Furthermore, some prehistoric plants look very alien to us, with long creeping axes unlike modern roots and strange blossom-like structures long before the evolution of flowers. It is useful to explore their forms, to analyse why they look entirely different from the forms we see today, and hypothesize what caused them to fail, since we no longer see them. Conversely, some extremely ancient plants look very similar to those we see today. It is also of interest to investigate what made these forms successful enough to last for hundreds of millions of years, through many radical environmental changes.

Palaeobotanists study prehistoric plant fossils around the world to answer some of these questions. One extremely well preserved and well studied site is the Rhynie chert, a 400-412 million year old deposit containing permineralized plant fossils. This site provides an important insight into plant evolution due to its age. The Rhynie chert plants are the descendants of the first land plants. They are the ancestors of the first leafy plants, as well as plants with specialized organs such as roots and seeds. Furthermore, the Rhynie chert plants are some of the first known vascular plants [80].

One tool for studying the development of plants in general and prehistoric plants in particular is computer modelling. Computer models are well suited to the task of analysing the development of plants as they can deal with the complexity of self-similar plant forms in ways that are easy to represent and manipulate. Growing plant structures can quickly
become very intricate, but their growth often follows simple, repetitive rules. Models can be used to simulate these rules and visualize the growth of a plant over time, which is important for prehistoric plants where the growth can no longer be observed in nature.

The models of Rhynie chert plants considered in this thesis have been created using ‘L-studio’ [61], an L-System based software platform for modelling plants [62]. ‘L-studio’ is a highly complex piece of software with its own scripting languages [38] and associated tools for defining L-systems. L-systems can represent branching structures as a simple string of symbols. Different symbols denote branch segments, rotation angles, organs and other elements of each plant. Simple string replacement is used to recursively replace a single branch segment with more complex forms. This may lead to plant models with complex structures. Stochastic productions and other conditions (such as collision detection between competing branches) can make the models look more natural and realistic, and respond to environmental influences.

1.1 Objectives

Many computer models of extant plants have been created over the years, and many techniques have been developed to create models effectively. A goal of this thesis work is to apply techniques devised for extant plants to the modelling of extinct plants, where much less information is available, and evaluate their suitability for this purpose. This was accomplished by examining the plants of the Rhynie chert.

Many previous reconstructions of extinct plants (e.g. [6] [34] [80]) simply present the static morphology of a plant observed in the fossil record. In contrast, I consider the development of extinct plants instead of just their static, mature morphology. I have used multiple fossil specimens of each plant species and the observations of palaeobotanists working with the fossils to reconstruct different stages of development of these plants, and model their growth. Focusing on development, as opposed to static morphologies observed in the fossil
record has the potential to yield a richer understanding of extinct terrestrial plants.

Environmental changes could be one of the biggest factors driving plants to evolve and diversify rapidly at different points in Earth’s history, usually following extinction events [29]. They are thus an important factor in understanding ancient plants, and are considered in my research. Environmental factors include terrain and climate, as well as competition with other plants for light, nutrients, and opportunities to spread seeds or spores. Environmental information can be found in the fossil record by examining the relative position and decay patterns of plants after their death and preservation [25]. The secondary goal of this thesis work is to reconstruct the environment of the Rhynie chert and the interactions between its plant species. Because the complexity of the ecosystem at the Rhynie chert is relatively small, it provides an opportunity to model the dynamics of an entire ancient ecosystem.

1.2 Organization

Chapter 2 contains background information, including a background of plant evolution, geologic setting, significance to science, and previous modelling literature regarding both plant and ecosystem modelling. Chapter 3 discusses the structure of different types of L-Systems, the formalism used in this thesis to model plants and their development, and the usage of L-studio and its components, as well as containing a glossary of important terminology.

Chapter 4 is split into seven sections, each describing one of the seven plant models capturing the seven species of higher plants present in the Rhynie chert. Each of these sections is further divided into four subsections. The first describes what is known about each species. The second gives an overview of how the model was created based on known information and assumptions. The third subsection give details on how the model was implemented in L-studio. The last section contains images showing the modelling results. The actual L-System implementations of the models are quite lengthy, and are included in
Chapter 5 explains how the environmental information regarding each species was incorporated to simulate the interactions between the plants and their environment. These interactions include the plant succession as the species re-establish themselves after a natural disaster such as a geyser eruption. Finally, Chapter 6 contains a summary and discussion of results, and points to topics for possible future work.
Chapter 2

Background

2.1 The Origin of Plants

2.1.1 The Evolution of Plants

Plants as we know them today began as photosynthetic cyanobacteria in the oceans of the Palaeoproterozoic era. The first multicellular antecedents to land plants belonged to the division Chlorophyta, or green algae. However, these do not occur until the Lower Palaeozoic. There is evidence of algal scum on land as far back as 1200 Ma, but these plants were not well adapted for life on land. The first adaptations to life on land seem to have appeared when simple aquatic plants colonized fresh water habitats. Because fresh water bodies can vary in size, and are sometimes only temporary, adaptations that allow a plant to survive out of the water become evolutionarily advantageous. The link between fresh water plants and land plants is poorly understood, because well-preserved prehistoric fresh-water algal specimens are rare compared to fossil terrestrial plants. However, the first plants to colonize land seem to be green algae with thick cell walls designed to help them survive in a desiccating environment. Specimens of Chlorophyta exist in the Rhynie chert.

2.1.2 Colonization of Land

The largest barrier that plants had to overcome in order to colonize land was the lack of readily available water. Most extant species of aquatic algae will die within minutes of being removed from the water. In contrast, although most land plants have a significantly larger surface area from which water can evaporate, they have evolved a plethora of adaptations to survive on land. Cutan is a substance that covers the aerial surfaces of some plants, and greatly reduces the rate of evaporation by creating a waxy layer that is impermeable
to water. Cutan has been found in the cuticles of the Rhynie chert plants \[80\]. In addition to retaining water, plants must also absorb water. Evidence of absorptive, rhizome-like anchoring cells occurs in ancient Chlorophyta, and the Rhynie chert plants seem to have well developed subterranean rhizomatous axes, or decumbent axes with absorptive rhizoidal hairs. Another source of water loss for plants is transpiration through their leaves and other surfaces. Modern plants have guard cells flanking their stomata, which reduce water loss during respiration by opening and closing the stomata. The plants of the Rhynie chert have also been found to possess stomata with guard cells. They are positioned directly on the aerial axes, and sometimes on decumbent axes as well. Plant reproduction also needed to adapt to life on land. Reproductive cells needed to be better protected against dessication, but also required a new method of distribution, through air instead of water. The plants of the Rhynie chert are some of the first known plants to bear sporangia and spores protected by a layer of sporopollenin, which prevent the reproductive structures from losing excessive amounts of water.

Because there are still gaps in the fossil record connecting ancient aquatic plants to land plants, the ancestors of the first known land plants are hypothetical. Lignier suggested that Chlorophyta colonized land by specializing their basal portion for absorption, and their now aerial portion for photosynthesis, water retention, and spore dispersal \[47\]. Other botanists corroborated his theories over time, culminating in Zimmerman’s Telome Theory \[89\].

Higher aquatic plants, such as those in the families Nymphaeaceae and Ceratophyllaceae, are thought to be descended from plant species that first adapted to life on land, then moved back to aquatic environments \[77\].

2.1.3 Evolution of Vasculature

The first confirmed vascular plant, predating the Rhynie chert flora, is \textit{Cooksonia} \[46\]. The fossil record of \textit{Cooksonia}'s ancestors is extremely sparse. Several older plants have been suggested to be vascular plants, but their existence is not generally accepted. Evidence for
many of these hypothetical species comes in the form of microfossils, which could originate from sources other than land plants \[5\].

*Cooksonia* is similar to some of the plants found at Rhynie, being dichotomously branched and bearing terminal sporangia, and it is likely closely related to *Rhynia*. Well-preserved fossils have revealed the presence of vascular tissue in the aerial axes of *Cooksonia* \[20\]. Other species with vascular tissue, related to *Cooksonia*, and predating the Rhynie chert flora include *Uskiella* \[71\] and *Steganotheca* \[21\]. Unfortunately, these species are poorly preserved, and little is known about them.

### 2.2 Geologic Setting

The Rhynie chert is an Early Devonian age chert deposit containing extremely well preserved fossils of plants, animals and fungi. The Early Devonian epoch was characterized by warm temperatures (about 6°C warmer than the modern global average), and the lack of any glaciation. Due to a high concentration of atmospheric carbon dioxide, the temperature gradient between the poles and the equator was less than it is today \[35\].

At present, the chert is located near the village of Rhynie, Scotland. At the time of deposition, the region was much closer to the equator, at about 28° south \[80\] (Fig. 2.1), so it is assumed that the climate was quite warm with little seasonal temperature variation. It was situated in the southeastern section of a larger continent known as Laurussia.

The chert was deposited by ancient hot springs, which coated plants and animals with mineral-rich sinters, preserving them almost perfectly \[69\]. Among the species preserved were at least seven different higher land plants, which represent some of the first species of plant to colonize land. The Rhynie chert is the oldest known hot spring system in the world, and would have been similar to modern hot spring systems like those in Yellowstone National Park, USA, and Deildartunguhver in Reykholtsdalur, Iceland. Some hot spring features, including surface features such as geyser vents, have been preserved at Rhynie \[80\].
Figure 2.1: A. Estimated palaeogeographic location of the Rhynie Chert. Image modified after [7]. B., C. Modern day location of the Rhynie Chert. Images courtesy of NASA.
2.3 Significance to Science

The preserved plants of the Rhynie chert represent a snapshot in time when some of the most interesting evolutionary processes in plant architecture were taking place. The Early Devonian saw the evolution of the first vascular plants and the first plants with sporangia, many of which are extremely well preserved at Rhynie. Shortly after this time, the first specialized leaves, roots and stems evolved, as well as the first seed plants, making the Rhynie chert flora their direct predecessors. The exceptional preservation of these plants during such an important period of plant evolution gives researchers a detailed picture of the morphology and internal structure of early land plants. The Rhynie chert flora also gives us an idea of how early plants first adapted to terrestrial environments, and what evolutionary pathways could have led from these simple plants to modern ones.

In addition to having a profound palaeobotanical significance, the Rhynie chert also holds discoveries significant to geologists. The preserved geysers at the Rhynie chert are the oldest known, and contain preserved surface features, which is rare at any time period. The preservation of ancient plants and animals in this system also gives geologists insights into how siliceous hot spring deposits can preserve organic remains so exceptionally [80].

2.4 Previous Modelling Work

2.4.1 Plant Models

The visual reconstruction of extinct plants and other natural forms is an important field. Extinct species appear only in the fossil record, and because the fossilization process is imperfect under the most favourable conditions, reconstruction is the only way to get an idea of what the plant looked like when it was alive. Models allow us to better understand and describe a plants architecture, and possibly evolutionary connections [56]. Before computer reconstruction became available, most reconstructions were either two-dimensional diagrams
(e.g. [43] [26]), or physical three-dimensional models (e.g. [80]). Computer models are more powerful because they can be used to reconstruct natural forms algorithmically or dynamically.

Ancient plants such as those at Rhynie have been studied and modelled conventionally for some time. The Rhynie chert was discovered by Dr William Mackie in 1912 [51], and by 1921, the first hand-drawn reconstructions were produced by Kidston and Lang [43]. Since then, many other hand-drawn reconstructions have been attempted by various authors. Examples of these are listed by plant species in Chapter 4. More recently, several three-dimensional physical models have been created [80], but to date, no computer-generated model of the Rhynie chert flora has been attempted.

One of the first algorithmic computer models was a reconstruction of extant shelled molluscs by Raup in 1966 [64]. He asserted that computer models could be used to create different forms by varying parameters. He used this technique to model various shell structures, which would have taken much more effort with conventional modelling methods, and was also able to simulate theoretically possible forms that do not occur in nature. Subsequent studies, for example by Cortie [14] [15] expanded on these by modelling more complex shells with more powerful computers.

Computer models of extinct species outside the Rhynie chert have existed for some time. A 1982 paper by McKinney and Raup entitled "A turn in the right direction: simulation of erect spiral growth in the bryozoans Archimedes and Bugula" [52] presents algorithmic reconstructions of bryozoans - plant-like animals with spiral and branching structures. A set of growth parameters were varied to simulate similar species with different morphologies.

Also in 1982, Niklas published a paper entitled "Computer simulations of early land plant branching morphologies: canalization of patterns during evolution?" [54]. Rather than modelling a specific plant, he analysed the evolution of different branching structures observed in early land plants. Random branching patterns were generated, then 'evolved',
and the results were compared to a lineage of fossil plants, which exhibited a progression of similar branching morphologies. Niklas’ simulations include a ‘pseudorhizomatous’ branching pattern similar to that observed in *Aglaophyton*. In his model, these axes have a limited capacity for vertical growth and progressively fall onto the substrate under their own weight, resulting in a low-growing plant which covers a large area.

In another paper, entitled "Mechanical and photosynthetic constraints on the evolution of plant shape" [56], Niklas and Kerchner generate different simple branching structures and analyse their fitness in terms of photosynthetic efficiency and mechanical stress based on their shape. The more successful morphologies were found to be similar to some extinct bryophytes, mosses, and tracheophytes. A subsequent paper, "Adaptive walks through fitness landscapes for early vascular land plants" [55], Niklas expands the parameters from the previous model to include reproductive success and minimized surface area (to reduce water loss). The most successful morphologies were again found to emulate the forms of some ancient vascular plants. The simulations also suggested that the necessity to conserve water was the tightest constraint on the diversification of early land plants.

A series of papers by Daviero *et al.* entitled "Computer simulation of sphenopsid architecture" [17] [16] used a modelling system created by AMAP [32] to digitally model extinct sphenopsid plants (a class of ancient, horsetail-like plants). They extrapolated descriptions of components and parameters of the plant being modelled from fossil evidence, and used the extant species *Equisetum telmateia* to fill in details to compensate for a limited number of fossil specimens. Their final models represent sphenopsids during different stages of growth.

In their 2006 paper "Evolution of Land Plant Architecture: Beyond the Telome Theory" [73], Stein and Boyer expanded on Zimmerman’s Telome Theory [89] by modelling various growth processes in generic ancient plants. Specifically, they aimed at creating a more rigid, testable version of the Telome Theory by modelling growth processes with concrete rules and parameters. cpfg and L-studio were used to create these models.
Boyer’s 2008 PhD thesis titled "Testing the Telome concept: A modeling approach for understanding the growth of early vascular plants" contains several further relevant studies. In the chapter "A Morphometric Analysis of an Early Euphyllophyte, *Psilophyton dapsile*", Boyer used palaeobotanical techniques to analyse fossils of *Psilophyton*. He extracted parameters such as axis length and width, and branching angles. In the chapter "Modelling the Morphogenesis of Basal Euphyllophytes" he used this information to create a computer model of early euphyllophyte plants (a subdivision of vascular plants). Plant structures and organs were modelled as separate components similar to L-system modules. Growth was simulated iteratively, regulated by a supply of energy that was augmented by photosynthesis and consumed by growth. He then compared the direct findings from the fossils to the computer model. Using statistical analysis, he concluded that the model was a reasonable approximation of the fossil based on measured parameters. He then varied the model input parameters to create a morphospace of the plant’s possible ancestors.

### 2.4.2 Ecosystem Models

Examples of previous work in the field of environmental modelling are diverse in their approaches, and mostly deal with extant or recent ecosystems. Previous modelling efforts of extinct ecosystems are almost non-existent.

A paper published in 1979 by Shugart *et al.*, titled "Environmental Gradients in a Simulation Model of a Beech-Yellow-Poplar Stand" is an early example of environmental modelling using global parameters. Growth and death of a stand of trees is abstractly modelled by representing their height, maximum diameter, and total biomass under different conditions. Small changes in average temperature were found to have a significant effect on stand growth.

A 1980 paper by Solomon *et al.* entitled "Testing a Simulation Model for Reconstruction of Prehistoric Forest-Stand Dynamics" is one of the rare published studies that deals with fossil data. This study used information such as climate stability, species immigration,
fire frequency, and disease to reconstruct forest dynamics over the last 16000 years. This information was determined by analysing the distribution and properties of preserved pollen. The quantity and distribution of trees over time was modelled.

Another 1980 paper, entitled "A Model for Growth and Self-thinning in Even-aged Monocultures of Plants" [2], was published by Aikman and Watkinson. This paper describes a model of the behaviour and survival rate of monocultured plants growing in close proximity. Each plant had an area and growth rate. The growth rate of a plant was impeded when another plant encroaches on its area. A negative growth rate caused the plant to die. This simulation provided realistic self-thinning behaviour when compared with published data. This concept was expanded upon by Firbank with Watkinson in their paper "A Model of Interference Within Plant Monocultures" [30]. In addition to modelling self-thinning, this model removed the even-aged property, giving an advantage to older, larger plants that can crowd out smaller plants and exploit more resources. Once again, the results were consistent with published data.

A report by Deutschman et al., entitled "Scaling from Trees to Forests: Analysis of a Complex Simulation Model" [19], described a detailed forest simulation. Nine different species compete for space in this model, and their success is determined by their access to light, which affects their growth rate, which in turn affects their mortality and reproduction rates. Different species grow, occupy space, disperse, and die differently depending on simulation parameters specific to each species. Successions were observed over long periods of time, and abiotic stresses were found to have an effect on which species dominated.

"Modeling and Visualization of symmetric and asymmetric plant competition" [3] is a 2005 paper by Alsweis and Deussen, which discusses unequal competition between plants due to differences in size or species. Each plant has a circular zone of influence which represents the distance at which it interacts with other plants. The radius is more dependent on the area the plant requires to provide itself with resources than its physical size. Plants
dominated by larger plants encroaching on their zone of influence are deprived of resources and therefore grow more slowly. The appropriate interactions between plants of different sizes create simulations with a high degree of realism.

Until recently, the development of most ecosystem models relied mostly on plant competition within a relatively uniform environment. A 1998 paper by Deussen et al. entitled "Realistic modeling and rendering of plant ecosystems" distributed plants more realistically by creating a water map on the terrain. This was accomplished by dropping water onto the terrain from above, and allowing it to be absorbed into the soil as it runs downhill. This results in drier high altitudes and swammier, low lying regions. Ch’ng’s 2007 paper, "Modelling the Adaptability of Biological Systems" introduces a parametrization of the preferences of many plants. Their preference is indicated by a number between zero and one. For example, a plant with a hydrology/moisture content preference of 0-0.34 is a desert plant. A preference of 0.8-0.89 indicates, saturated, swampy soil, and 0.96-1.0 is the preference for immersed aquatic plants. In addition to moisture preferences, the paper also describes a parametrization for mean temperature preference, soil depth, slope conditions, and soil acidity and texture. While most recent works use some form of zone of influence to determine plant crowding, this paper also models adaptation to crowded spaces as a parameter. A crowding level in the range 0-0.35 is preferred, while higher levels of crowding will have a detrimental effect on the plant’s success.
Chapter 3

Modelling Methods

3.1 L-Systems

The Rhynie Chert plants were small, herbaceous plants with recursive branching structures. L-systems (or Lindenmayer systems) are string rewriting grammars well suited for modelling plants of this nature [60]. A basic L-system consists of an alphabet of characters or symbols, which can be arranged in a string, a list of productions dictating which symbols can be replaced and in what way, and an ‘axiom’, specifying the initial string before any replacements are made. The string then undergoes a series of derivation steps, wherein production rules are applied to all characters in the string simultaneously (Fig. 3.1). This differentiates L-systems from other string rewriting systems, and allows them to better represent physical phenomena where processes occur concurrently. If there is at most one production for each symbol in the alphabet, the L-system is deterministic. However, it is possible to have multiple production rules for the same symbol, with different weights (propensities), producing a stochastic L-system (Fig. 3.4).

A geometric interpretation of the symbols is also needed. The most common geometric interpretation of L-systems is based on ‘turtle geometry’ [1]. The ‘turtle’ is essentially a reference frame in space, and the string serves as an instruction set which determines the turtle’s motion. A symbol can be interpreted as ‘move forward’ (e.g. F in Fig. 3.1), or ‘rotate the turtle about an axis’ (e.g. + in Fig. 3.1). By interpreting the entire string from left to right in this fashion, a graphical representation is produced.

Two special characters, denoted by square brackets ([ and ]), are included in the alphabet to denote branching structures. When interpreting the string geometrically, an opening bracket pushes the current position and orientation of the turtle to a stack, and a closing
Figure 3.1: An example of a basic L-system which produces a simple tree-like branching structure.

Axiom: $A$

- $A ightarrow F(+A)/A$
- $F ightarrow F+F-F-FF+F+F-F$ (Rotate 45°, Flip Horizontal)

1: $F(+A)/A$
2: $F[+F(+A)/A]/F[+A]/A$
3: ...

Figure 3.2: The quadratic Koch curve at iterations 1, 2 and 4. This fractal can be generated by an L-system with only one production rule, $F ightarrow F+F-F-FF+F+F-F$. 
Figure 3.3: An example of a parametric L-system, which produces a tree-like branching structure with organs and internodes of varying size.

bracket pops from the stack, returning the turtle to the location of the last opening bracket. This is essential for replicating the branching structures of biological forms.

Other special symbols are sometimes used to interpret L-system strings. Commonly, # is used to denote changes in line width, and ; is used to denote changes in colour.

Replacing a symbol with a substring that contains the original symbol leads to self-similarity after multiple iterations, and therefore a simple set of production rule can produce complex fractal forms (Fig. 3.2). L-systems are well suited to modelling plants due to their recursive nature, which corresponds to the fractal aspects of many plant forms.

3.1.1 Parametric L-Systems

Basic L-systems are capable of producing fractals and simple biological forms, but more control is often needed. Parametric L-systems allow each symbol in the alphabet to be
associated with one or more parameters \cite{33}. These parameters can be used to represent 
continuously varying quantities such as the length, width or overall scale of an organ, or 
the magnitude of a branching angle. Parameters can be varied over time using arithmetic 
expressions in the production rules (Fig. \ref{fig:complexity}).

3.1.2 Context Sensitive L-Systems

Context sensitive L-systems add another level of complexity and versatility to the basic L-
system formulation. In context sensitive L-systems, symbols adjacent to the symbol currently 
being considered can influence which production is applied. This allows different components 
of the model to interact with each other, greatly increasing the power of these models for 
biological modelling applications (Fig. \ref{fig:context}).

3.1.3 Environmentally Sensitive L-Systems

Environmentally sensitive L-systems are capable of interacting with external factors outside 
the L-system string \cite{59}. This is often essential for the modelling of plant life since it can be 
used to simulate interactions between the plant and the terrain, sources of light, water and 
nutrients, and other plants (Fig. \ref{fig:environment}).

Two special productions can be used to gather environmental information. The ‘position’ 
symbol has three parameters, which are updated with the turtle’s $x$, $y$, and $z$ coordinates 
during each simulation step. This symbol can be used to change the L-systems’ behaviour 
based on its position in space. For example, it can detect collisions with a heightmap by 
checking whether the turtle is located below the surface. The ‘heading’ symbol also has 
three parameters, which are updated with the $x$, $y$, and $z$ components of the turtle’s heading 
vector during each simulation step. This can be used to modify a plant’s growth behaviour 
depending on the orientation of its branches.

Plant interactions with water and soil are discussed further in chapter \ref{chapter:water}.
Figure 3.4: An example of a context sensitive, stochastic L-system, which produces a tree-like branching structure with a more organic feel. Both sympodial and monopodial branching is possible via two productions for A. They will be selected with probabilities $\frac{3}{4}$ and $\frac{1}{4}$, respectively. The productions for B will be selected based on whether or not $A(y)$ or $[B(xR,wR)][B(xL,wL)]$ exists in its right context. In the latter case, B modules preceding a branching point will increase in width according to the pipe model.
3.2 L-studio

L-studio is a programming environment designed to model individual plants or organisms, or entire ecosystems, using L-systems [61]. In addition to allowing users to write, compile and run L-system-based models, L-studio contains various tools to make the modelling process easier and more powerful. These will be described in the following sections.

3.2.1 cpfg and lpfg

Two L-system-based simulation programs are available to users of L-studio. cpfg (continuous-parameter plant and fractal generator) expresses L-systems with a mathematical notation, using a list of productions which implicitly define the alphabet [53]. Some symbols (also known as modules) are reserved for operations specific to the program, such as setting the width or colour of line segments, or drawing a circle or sphere (Table 3.1). Arithmetic operations can be performed as part of the productions. In addition to the list of productions
which are used every derivation step, there is a list of ‘homomorphism’ productions which are applied to the string after each step, before rendering. These allow the user to translate a simple symbol into a more complex representation. For example, the symbol $L$ could be translated into a model of a leaf. A symbol representing a line in a fractal or a plant’s axis can be interpreted as a 2D line or a generalized cylinder.

$lpg$ is similar to $cpfg$, but instead of expressing L-systems formally, it builds on C++ syntax to give the user much more power over the modelling process [37]. C++ code can be executed as part of the derivation process. Productions allow the user to preform tasks such as arithmetic operations, conditional statements and loops in C++ syntax before concluding with the ‘produce’ statement (Fig. 3.7). $lpg$ modules and their parameters must be defined at the beginning of the program. Modules will be denoted by typewriter font in subsequent sections (E.g. Apex).

Both $cpfg$ and $lpg$ employ OpenGL [70] to visualize the simulation results. Textures can be included in the final rendering, and light position, viewing projection and many other
module A(float);
module B(float,float);

derivation length: 10;
Axiom: StartGC() A(1);

production:
A(x): {
    float random = ran(1);
    if (random < 0.75) {
        produce B(0.2,0.1) RollR(180) SB() Left(45) A(x) EB()
        SB() Right(16) A(x+1) EB();
    } else {
        produce B(0.2,0.1) RollR(180)
        SB() Left(34) A(x) EB() A(x+1);
    }
}

consider: A;
B(x,w) >> A(xL): {
    produce B(x*(1.0+xL/10.0), w);
}

consider: B;
B(x,w) >> SB() B(xR,wR) EB() SB() B(xL,wL) EB(): {
    produce B(x*1.33, sqrt(pow(wR,2.0f) + pow(wL,2.0f)));
}
B(x,w) >> SB() B(xR,wR) EB() B(xL,wL): {
    produce B(x*1.33, sqrt(pow(wR,2.0f) + pow(wL,2.0f)));
}
B(x,w): {
    produce B(x*1.33, w);
}

interpretation:
A(x): {
    produce F(0.6) F(0) SetColor(2) Sphere(x/10) EndGC();
}
B(x,w): {
    produce SetWidth(w) F(x) PointGC();
}

Figure 3.7: An example of an L-system written in lpfg. This L-system is equivalent to the cpfg-based example shown in Fig. 3.4. The consider statements specify which modules are noticed when using context-sensitive productions.
<table>
<thead>
<tr>
<th>Parameters</th>
<th>cpfg</th>
<th>lpfg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Branch</td>
<td>SB()</td>
<td></td>
</tr>
<tr>
<td>End Branch</td>
<td>EB()</td>
<td></td>
</tr>
<tr>
<td>Turtle Forward</td>
<td>F(x)</td>
<td>F(x)</td>
</tr>
<tr>
<td>Rotate Left</td>
<td>+(...θ)</td>
<td>Left(θ)</td>
</tr>
<tr>
<td>Rotate Right</td>
<td>-(...θ)</td>
<td>Right(θ)</td>
</tr>
<tr>
<td>Rotate Up</td>
<td>^(...θ)</td>
<td>Up(θ)</td>
</tr>
<tr>
<td>Rotate Down</td>
<td>&amp;(...θ)</td>
<td>Down(θ)</td>
</tr>
<tr>
<td>Roll Left</td>
<td>(...θ)</td>
<td>RollL(θ)</td>
</tr>
<tr>
<td>Roll Right</td>
<td>/(...θ)</td>
<td>RollR(θ)</td>
</tr>
<tr>
<td>Set Colour</td>
<td>;(i)</td>
<td>SetColor(i)</td>
</tr>
<tr>
<td>Set Contour</td>
<td>@#(i)</td>
<td>CurrentContour(i)</td>
</tr>
<tr>
<td>Set Width</td>
<td>#(w)</td>
<td>SetWidth(w)</td>
</tr>
<tr>
<td>Draw Circle</td>
<td>@o(d)</td>
<td>Circle(r)</td>
</tr>
<tr>
<td>Draw Sphere</td>
<td>@O(d)</td>
<td>Sphere(r)</td>
</tr>
<tr>
<td>Get Position</td>
<td>?P(x,y,z)</td>
<td>GetPos(x,y,z)</td>
</tr>
<tr>
<td>Get Heading</td>
<td>?H(x,y,z)</td>
<td>GetHead(x,y,z)</td>
</tr>
<tr>
<td>Start Generalized Cylinder</td>
<td>@Gs</td>
<td>StartGC()</td>
</tr>
<tr>
<td>Generalized Cylinder Point</td>
<td>@Gc</td>
<td>PointGC()</td>
</tr>
<tr>
<td>End Generalized Cylinder</td>
<td>@Ge</td>
<td>EndGC()</td>
</tr>
</tbody>
</table>

Table 3.1: Important symbols and modules in cpfg and lpfg.

rendering parameters can be set outside the L-system definition.

For this thesis work, cpfg was used for early modelling efforts and experimentation. All of the final models were implemented using lpfg.

3.2.2 Functions, Contours and Surfaces

L-studio allows the user to control the structure and appearance of their models using functions, contours and surfaces. Functions are defined by constrained open B-spline curves\[1\] with a domain of $[0, 1]$. The domain and range of these functions can be modified from within the L-system code if necessary. Functions are useful for a variety of tasks, such as modelling smooth growth over time, changing decay rates, or specifying longitudinal contours of plant organs.

\[1\]Each successive control point’s x coordinate must not decrease to prevent the curve from violating the definition of a mathematical function.
Contours in L-studio are also modelled as B-spline curves. These can be open or closed curves, with no constraints on domain, range, or relative control point location. Closed contours are useful for modelling transverse sections of plant axes or organs, while open contours can be used to model leaf surfaces, petals, or, in the case of this work, sporangial valves. By default, a circular contour is used for all axes, but this can be replaced by a user-defined contour at any time.

Finally, surfaces can be used to model entire plant organs or other objects. This is accomplished using one or more B-spline patches. Once a surface has been defined, it can be rendered into the model at any time during the interpretation.
3.2.3 Tropism and Elasticity

Many plants exhibit phototropisms or gravotropisms, giving their axes a distinctive curved shape which needs to be captured in the models. Both cpfg and lpfg allow the user to define a general tropism for a plant by specifying a direction and strength value, known as elasticity. Each time the turtle moves, a vector based on the tropism direction and elasticity is added to its final position. Thus, with an upward direction and a positive elasticity value, the axes of a modelled plant will curve upward over multiple segments.

3.3 Terminology

A significant amount of biological and geological terminology is used in this thesis, which may not be familiar to some readers. Furthermore, some of these terms have special implications in the context of L-systems. Terms of interest are discussed in this section.

**Adventitious** branches are shoots that emerge from an unusual position.

An **apex**, or shoot apical meristem, is the growing tip of a plant. Apices add new cells to the tips of shoots and roots in extant plants. In L-systems, apices are usually the module at the tip of each axis (A) which is responsible for adding new internodes and determining branching points.

The term **axis** refers to the main stems of a plant. In L-system terms, an axis is a sequence of internodes tipped by an apex.

The **branching angle** of a plant is the angle between two branches or the angle between a side branch and the main axis. The branching angles of plants in the Rhynie chert are assumed to be uniform across the whole plant. In L-systems, branching angles are represented by rotations to the turtle (E.g. \(+35\)).

**Chert** is a type of dark, hard rock with a microscopically fine-grained texture. It often occurs as nodules, and massive chert beds like those at Rhynie are less common.

**Circinate vernation** is a process by which plants protect their fertile axes by growing
in a coiled position until they are mature. This is common in extant ferns, which produce
fiddleheads that unfurl when they reach maturity.

Plant colonization is the process by which plant populations arise in formerly barren
habitats. Plant colonization and recolonization was common at the site of the Rhynie chert
due to geyser activity. Plant populations could be wiped out (and preserved) by a geyser
eruption, and new plants would colonize the newly deposited substrates later.

Corms are short, bulbous, subterranean plant stems that often serve as storage organs
for plants that face adverse conditions such as cold winters and droughts.

The cortex of a plant is its outermost layer of cells.

Decumbent axes are those which lie close to or directly on the ground, turning upwards
locally into aerial axes.

Dehiscence is the process by which a sporangium splits into two halves in order to
spread its spores once it has matured. These sporangia contain a built in point of weakness
where dehiscence will occur, which is visible in fossils.

A dichotomous branch is a branching point at which the plant’s main axis splits into
two equal axes, diverging from the main axis at the same angle in either direction. In
contrast, sympodial branches consist of two new apices, while the original apex forms a bud
or dies. The L-system construction of a dichotomous branch is $I[++A][-A]$.

An enation is an outgrowth on the surface of a plant organ.

Generalized cylinders are cylinder-like structures whose cross sections can be any curve,
rather than a circle.

An internode is the part of a plant’s stem between two nodes. In the case of the models
presented in this thesis, it is the section of the plant between two branching points. For
modelling purposes, internodes are further divided into internode segments (E.g. $III...$).

Plant litter is a substrate composed of decaying parts of dead plants. Many plants prefer
to grow on plant litter since nutrients from the dead plants are readily accessible.
A longitudinal section of a fossil is a cut along its long axis, to reveal its interior structure. A longitudinal section is the opposite of a transverse section.

A monopodial branch emerges at an angle from the main axis of the plant, while the main axis continues in its original direction. The L-system construction of a monopodial branch is $1[+A]A$.

A monotypic stand of plants is a stand where only one species occurs. This could happen because the environment is only suited to one species, or because that species is an early colonizer.

Orthotropism refers to the behaviour of plants to grow upwards, either to grow towards a source of light, or to oppose gravity.

The pipe model is a method of determining the diameter of a branch based on the diameter of its child branches. Ideally, the width of any branch is $\sqrt{d_r^2 + d_l^2}$, where $d_r$ and $d_l$ are the diameters of the right and left child branches. In reality, this relationship can sometimes be expressed with exponents other than two [50].

The phyllotactic divergence angle of a plant is the angle between organs or branches at each consecutive node. The most common phyllotactic divergence angle in extant plants is 137.5 degrees, which seems to be reflected in ancient plants as well. In L-systems, phyllotactic divergence angles are represented by rolling the turtle (E.g. /$(137.5)$).

The plant succession refers to the process by which the composition of a plant community changes over time. Early colonizers will begin the community before being replaced by slower growing, more competitive species. In the case of the Rhynie chert, the plant succession process began frequently due to geyser eruptions that wiped out populations.

A plastochron is the time interval between two successive recurring events in a plant’s growth process. In these models, it refers to the growth period between two successive branching points.

Pseudomonopodial branching is similar to monopodial branching, but differs in how
it arises. Two branches are formed but one is deflected by a negligible amount, or not at all. The L-system construction of a pseudomonopodial branch is I [+A] [A].

**Rhizoids** are tiny hair-like structures that appear on the undersides of the roots or rhizomes of some plants. They can serve to anchor the plant and conduct water.

**Rhizomes** are continuously growing horizontal underground stems that give rise to intermittent aerial shoots and adventitious roots.

**Sinter** is a type of mineral deposit found in hydrothermal areas. It is formed by precipitation from silica-rich water erupted from geysers or hot springs.

**Sporangia** are the organs of fungi and some plants where spores are produced.

If an axis of a plant is **subaerial**, it grows above ground, or ‘under the air’.

The term **substrate** refers to the composition of the ground on which a plant grows, which could be soil, sand, litter, sinter etc.

**A transverse section**, or cross-section of a fossil, is a cut perpendicular to its main axis which exposes the interior. A transverse section is the opposite of a longitudinal section.

**A sporangial valve** refers to one half of a dehiscent sporangium.
Chapter 4

Plants of the Rhynie Chert

4.1 General Information

To date, seven distinct species of macroplant have been identified from fossils in the Rhynie chert beds. These fossils are extremely well preserved, allowing for the analysis of both their morphology and their internal cellular structure. The Rhynie chert plants have been named, in alphabetical order, *Aglaophyton major*, *Asteroxylon mackiei*, *Horneophyton lignieri*, *Nothia aphylla*, *Rhynia gwynne-vaughanii*, *Trichopherophyton teuchansii*, and *Ven.tarura lyonii*. They are usually identified by their genera. Five of these have been confirmed as vascular plants. *Aglaophyton* and *Nothia* possess some features of modern vascular plants, but are not considered true vascular plants because they lack tracheids. Except for *Asteroxylon*, which is a member of the extant division *Lycophyta*, none have any living relatives.

4.2 General Modelling Techniques

Each plant begins as a *spore* module, which is randomly placed on the terrain. If the spore lands on a suitable substrate which is not submerged, it begins to grow depending on the topography of terrain around it, with the productions and parameters of its species. The tip of each axis is a shoot apex module or a rhizome apex module. A shoot apex module increases the length of the terminal axes over time, determines when and where new branches arise, and eventually may produce sporangia. A rhizome apex performs similar functions, but it cannot produce sporangia. Instead, it acts as a sensor that tells the rhizomatous axis how to react to the terrain. The behaviour of both is dictated by a parameter referred to as ‘vigour’. As the plant grows, the vigour of each apex decreases, and the behaviour changes as the
Figure 4.1: A diagram of a generic Rhynie plant. The red lines indicate the location of the ground depending on whether the plant’s horizontal axes are subterranean or subaerial. Sections of the plant are labelled as follows: A shoot apices. I aerial internodes. S possible sporangium-forming region (lateral or terminal). AR rhizome apex. IR rhizome internode. B plant base (spore location). $t_t$ location of termination threshold in terms of apex vigour. $t_b$ location of branching threshold in terms of apex vigour.

Vigour passes one of two thresholds. The first threshold is the branching threshold. Below this threshold, the plant can form new branches. The second is the termination threshold. Once the vigour of an apex reaches this threshold it terminates. Sporangia can usually be produced between the two thresholds.

The main section of each axis is composed of internodes and rhizome internodes. These represent internal sections of the axes between branching points or below the apex. They are produced at a predetermined initial width by the apices as the plant grows, and thicken over time in accordance with the pipe model [68]. Rhizomatous axes are almost identical to the aerial axes, with the exception that they usually grow horizontally, and can detect the terrain surface and settle onto it. Each internode is composed of ten or more internode segments in the models. This allows internodes to bend smoothly, achieving an organic appearance.

There are separate modules for sporangia, or groups of sporangia, depending on the nature of the sporangia of the plant in question. These modules do not affect the branching architecture of the model after they are created. Some plants bear their own unique features,
which will be described in the appropriate sections.

In addition to structural modules, the models include some invisible modules used to regulate the plant’s behaviour. The plants with subaerial decumbent axes have sensor nodes associated with their internodes, which detect the position of the ground. The rhizomatous sections also contain bend nodes, which also bend the rhizomatous axes downward until they settle on the substrate. All plants also contain an invisible Base node at the location of the initial spore, which stores the plant’s position and total age. This is used in ecosystem simulation to determine how long a plant will live and how it will interact with its neighbours. The growth period of these small plants is assumed to be short, and thus relatively unaffected by non-catastrophic environmental changes. This is difficult to determine, however, since the latitude of the chert and Earth’s mean temperature at the time indicate that there may not have been significant seasonal changes.

For all plants, one unit in the model is equal to one centimetre in real space.

Many of the plants share the same modelling parameters. Four of the most important parameters common to all plants are detailed below.

Every model specifies the initial width of internode segments when they are generated, in centimetres. This is handled separately for rhizomatous axes of the same plants, which
may have different dimensions. It directly impacts the final width of the axes, for which data exist, and need to be accurately reproduced in the model. The branching angle of each species is defined in degrees, and is assumed to be normally distributed around the average branching angle observed in the fossil record to avoid a rigid, artificial appearance. The phyllotactic divergence angle between consecutive branching points for each model is also specified in degrees. This is perturbed by a normal distribution to make the result more visually appealing. The phyllotactic angle is usually assumed to be 137.5 degrees based on observations of some fossil samples (Fig 4.2), and modern plants. Finally, the initial ‘vigour’ value for the rhizomatous or aerial apices is defined for each species. Vigour is an arbitrary value normally distributed with a mean of close to one\(^1\), which decreases as the plant grows. The behaviour of the plant is dependent on its current vigour value. Each apex has its own vigour value, which decreases over time, or as a result of other conditions depending on the specific model.

All models are implemented to be view dependent. Each model has some features that are only rendered in detail when the camera is pointing at them, and when the zoom is beyond a pre-defined threshold. Internodes and other features are not rendered when they are outside the view volume. This was implemented to allow for a greater level of detail to be rendered without compromising performance due to high polygon counts.

Some of the more simplistic models were used to experiment with the modelling of plant death. To accomplish this, the plant starts to shrivel up when its age (recorded by the Base module) reaches a predefined threshold. The angle of the plant’s internal bend nodes is then exaggerated slowly, causing the plant to twist and bend as if it had died of dehydration. Any part of the plant that comes into contact with the ground breaks off (Fig. 4.3A). Another version of plant death was also implemented, which attempted to simulate a plant dying from other conditions such as disease, soil depletion, or over hydration. In that case, each of the plant’s axes droop downward and become discoloured (Fig. 4.3B).

---

\(^1\)This varies by species.
Figure 4.3:  
A. Death of *Horneophyton* by shrivelling. The existing bends within the axes are exaggerated over time.  
B. Death of *Horneophyton* by withering. All axes bend downwards.

4.3 *Horneophyton lignieri*

4.3.1 Description

*Horneophyton* is the most commonly found fossil plant with subterranean axes, and one of the most common plants in the Rhynie Chert. It featured cylindrical aerial axes that branched dichotomously and repeatedly, bending upwards after the branching points. Most axes achieved a maximum width between 1.5 mm and 2.0 mm. The plant as a whole could reach a maximum height of about 20 cm. Its most unique features are its rhizomatous axes and sporangia. The rhizomatous axes were primarily subterranean, although they may have protruded from the soil (Fig. 4.4). They were tuberous, and bore rhizoid-like structures on their lower surface. The maximum observed width of a *Horneophyton* rhizomatous axis is 5 mm \[41\]. The sporangia of *Horneophyton* were terminal and had a unique multi-lobed shape, which arose from dichotomies within the sporangia themselves [26]. The sporangia have been observed to dichotomize up to three times, producing a structure with two to five lobes [28] (Fig. 4.5). They were approximately 7.5 mm in length.

*Horneophyton’s* preferred living conditions were damp to wet environments, determined by the association between *Horneophyton* fossils and water-borne fungi [76]. *Horneophyton*
fossils in life position are often found in monotypic stands, suggesting that it was a colonizer of sinter substrates. In other chert beds, it is found in association of other Rhynie plants, suggesting that it remained competitive with other species later in the plant succession [58].

Several previous reconstructions of *Horneophyton* are available, and are detailed in Table 4.1. However, *Horneophyton* reconstructions are chronically poor, largely because they have built off of each other rather than respecting fossil evidence, and thus errors have become exaggerated [81] [82]. These problems are commented upon in Table 4.1 as well.

4.3.2 Modelling

The main parameters used to model *Horneophyton* are listed in Table 4.2. Each cylindrical axis grows for a normally distributed period of time representing one plastochron, and then bifurcates. When an axis bifurcates, the branching angle is set at $X \sim N(44, 16^2)$ [2]. Because

---

2 Based on previous reconstructions.
<table>
<thead>
<tr>
<th>Reconstruction</th>
<th>Source</th>
<th>Type</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Diagram 1]</td>
<td>[4]</td>
<td>Illustration</td>
<td>Modelling of sporangia does not match fossil evidence [87].</td>
</tr>
<tr>
<td>![Diagram 2]</td>
<td>[26]</td>
<td>Illustration</td>
<td>Modelling of sporangia does not match fossil evidence [87], rhizomatous axes are too large [88].</td>
</tr>
<tr>
<td>![Diagram 3]</td>
<td>[74]</td>
<td>Illustration</td>
<td>Modelling of sporangia does not match fossil evidence [87], rhizomatous axes are too large [88].</td>
</tr>
<tr>
<td>![Diagram 4]</td>
<td>[81]</td>
<td>Model</td>
<td>Modelling of sporangia does not match fossil evidence [87] and the rhizomatous axes are too large [88]. The black colour of aerial axes is unjustified.</td>
</tr>
</tbody>
</table>

Table 4.1: Selected previous reconstructions of *Horneophyton*, and some of their drawbacks.
Figure 4.5: A. Comparison of a cross-section of a three-lobed *Horneophyton* sporangium to its counterpart in the model. B. Comparison of a cross-section of a two-lobed *Horneophyton* sporangium to its counterpart in the model. Red lines indicate dichotomies in the sporangia. Plates (left) modified from [26]. Images from models (right) edited to remove artefacts.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Source</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width of aerial axes</td>
<td>&lt;2 mm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Width of rhizomatous axes</td>
<td>&lt;5 mm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branching angle</td>
<td>$X \sim N(44, 16^2)^\circ$</td>
<td>Assumed</td>
<td>Based on previous reconstructions</td>
</tr>
<tr>
<td>Phyllotactic angle</td>
<td>$X \sim N(137.5, 5^2)^\circ$</td>
<td>Assumed</td>
<td>Based on phyllotactic angle of 137.5° observed in <em>Asteroxylon</em></td>
</tr>
<tr>
<td>Length of internodes</td>
<td>Variable</td>
<td>Assumed</td>
<td></td>
</tr>
<tr>
<td>Height of plant</td>
<td>~20 cm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dimensions of sporangia</td>
<td>5.0 mm x 7.5 mm</td>
<td></td>
<td>Variable, based on the number of lobes</td>
</tr>
</tbody>
</table>

Table 4.2: Values and sources of important modelling parameters for *Horneophyton*.

of the shape of its branches, it is assumed that *Horneophyton* is orthotropic. Each internode segment is deflected upwards by a small amount when it is produced, giving the axes their curved shape.

The behaviour of the apex is controlled by its individual vigour value. The plant may branch as long as the vigour is above the branching threshold, and once the vigour drops below that threshold, the axis has a chance to produce a sporangium during each time-step. If the axis fails to produce a sporangium before the vigour drops below the termination threshold, the axis terminates. To achieve the general shape of the plant proposed in previous reconstructions, vigour decays more rapidly when the axes are not vertical. This favours growth of the more upright axes, giving the plant a tall, slender shape. When axes are first produced, they are 0.5 mm in width. As they mature, they thicken and usually reach a maximum diameter of about 1.5 mm to 2.0 mm. Since the bases of the plants are usually poorly preserved or absent, the model allows slightly greater thicknesses to arise near the base of the plant. This liberty was taken to make the plants appear structurally sound, using extant plants as a guide.

Because of the simplicity of *Horneophyton*’s aerial axes, much of the modelling effort was directed towards the rhizomatous axes and sporangia. *Horneophyton* sporangia are generated at the ends of each fertile axis when the vigour level of the apex drops below
the predefined threshold, which indicates that the apex has reached maturity. The new sporangium then grows and bifurcates in much the same way as the aerial axes, but with much shorter internodes, and a different outer appearance. Most sporangia have a minimum of two lobes, and after three bifurcations, they can reach a total of eight lobes. Although five lobes is the maximum observed in fossils, eight is the theoretical limit after three bifurcations, and may (rarely) occur in the model.

The rhizomatous axes of *Horneophyton* are modelled as a bulbous, irregularly shaped structure below the substrate. Depending on the size and initial depth of the plant, the top of the axis may lie above ground. The axis’ maximum width is about 2.5 times the width of the base of the aerial axis, giving it an average width of 5 mm.

4.3.3 Implementation

The simulation begins with an axiom of one or more Germinator modules. Each Germinator has its initial delay decremented by one until it reaches zero, at which point it produces a (group of) *Horneophyton* Base and Rhizome module(s) at a random location on the terrain surface (Fig. 4.10). The position of the plant is stored as a parameter of Base. The Rhizome module is initialized with a random vigour value, and all other parameters zero. It grows a segment of length of 0.05 mm before selecting a contour (Fig. 4.6) and producing a RhizomeInterior module. Each RhizomeInterior module is initialized with an age of zero, identical length and vigour to its parent Rhizome, a predetermined width depending
Figure 4.7: The base functions used to define different aspects of Horneophyton’s form. A. growthRateF, used to smoothly grow each apex and internode. B. colourF, used to smoothly interpolate between different colours in the model. C. vigourDecayF, a probability distribution used to determine the percentage of an apex’s vigour that will remain after each time step. D. rhizomeWidthF, an approximation of the longitudinal section of Horneophyton’s rhizomatous axes.
Figure 4.8: Possible colours of *Horneophyton*, selected by the `colourF` function.

Figure 4.9: The contour lines used to give the sporangia of *Horneophyton* their irregular surface.

<table>
<thead>
<tr>
<th>Module</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germinator</td>
<td>delay</td>
<td>Places spore on the terrain, and tracks the delay before it germinates</td>
</tr>
<tr>
<td>Apex</td>
<td>age, length, vigour</td>
<td>Determines when and how the plant grows, branches, produces sporangia, and terminates</td>
</tr>
<tr>
<td>Internode</td>
<td>age, length, vigour, width</td>
<td>Represents internal sections of the plant, and determines their length and width</td>
</tr>
<tr>
<td>Sporangium</td>
<td>age, length, vigour</td>
<td>Determines growth and branching within the sporangium, similar to ‘Apex’</td>
</tr>
<tr>
<td>SporangiumNode</td>
<td>age, length, vigour, width</td>
<td>Represents internal sections of the sporangium, similar to ‘Internode’</td>
</tr>
<tr>
<td>Rhizome</td>
<td>age, length, numberOfNodes, vigour</td>
<td>Determines the growth of the rhizomatous axes, similar to ‘Apex’</td>
</tr>
<tr>
<td>RhizomeInterior</td>
<td>age, length, vigour, baseWidth, widthFactor</td>
<td>Represents the interior of the rhizomatous axes, similar to ‘Internode’</td>
</tr>
<tr>
<td>Base</td>
<td>age, position</td>
<td>Stores the total age and position of the plant</td>
</tr>
</tbody>
</table>

Table 4.3: A list of the L-System modules used to model *Horneophyton*. 
Figure 4.10: A diagram showing the structure of *Horneophyton*’s rhizomatous axes, including a simplified L-system interpretation. Sections of the rhizome are labelled as follows: $\text{Ir}(w,b)$ rhizome interior segments, with width factor and base width. $\text{Ar}(s)$ rhizome ‘apex’, with number of segments. $A$ aerial apex. $I(w)$ aerial internode, with width. $f(s)$ represents the ‘rhizomeWidth’ function (Table 4.4).
Function | Formula
--- | ---
growthRate(age) | $\text{growthRateF}(\text{age}) \cdot \text{vigour} \cdot (X \sim N(\mu, \sigma^2)) \cdot dt$
colour(age) | $\left[ C_1 + \left( (C_n - C_1) \cdot \text{colourF} \left( \frac{\text{age}}{\text{factor}} \right) \right) \right]
vigourDecay(yh) | $1 - \text{factor} \cdot \left[ 1 - (\text{vigourDecayF}(X \sim U([0, 1])) \cdot yh^{a \cdot b}) \right] \cdot dt$
rhizomeWidth(n) | $\text{scaleFactor} \left( \frac{\text{rhizomeWidthF}(n+1)}{\text{numSegments}} \right)$

Table 4.4: Functions used to model different aspects of *Horneophyton*. $\text{growthRate(age)}$ returns the amount that a section of the plant will increase in length, which slows over time. $\text{colour(age)}$ returns the index to a colour in the colourmap. $\text{vigourDecay(yh)}$ returns a factor by which the current vigour will be multiplied, which is dependent on the orientation of the apex. $\text{rhizomeWidth(n)}$ returns the width of ‘rhizome’ segment $n$. See Fig. 4.7 for base functions, and 4.8 for definition of $C_1$ and $C_n$. $a$, $b$, $c$, and $\text{factor}$ are constants.

on its vertical position, and an initially undefined ‘widthFactor’ (Table 4.3). Once thirteen RhizomeInterior modules have been produced, the Rhizome module is replaced by an Apex module. All its parameters are zero except for the predetermined vigour. During each subsequent simulation step, the RhizomeInterior modules age, increase in length according to the model’s ‘growthRate’ function (Table 4.4), and inherit the value ‘widthFactor’ from the first Internode at the base of the plant. The width of each RhizomeInterior module is a function of its ‘widthFactor’ and ‘rhizomeWidth’ (Table 4.4). Together, these give the largest part of the rhizomatous axis a width of approximately 2.5 times the width of the plant base.

Once the Apex has been produced, it ages by a factor of $dt$ each time step, grows in length according to the ‘growthRate’ function, and the vigour begins to decay in accordance with the ‘vigourDecay’ function. If the vigour parameter drops below the termination threshold, the Apex module is removed, terminating the axis. A Sporangium can be produced if the following conditions are met: the vigour falls within a fixed range, the age of the apex is greater than the termination threshold, and a random check is passed (15 %). When the age of the apex reaches a higher fixed threshold, the axis is rolled by one twelfth of the plant’s

---

3Arbitrarily selected to give the rhizomatous axis enough vertical subdivisions to be smooth.

4This prevents all sporangia from appearing the same distance along a terminal axis, which would make the plant appear too artificial.

5Such that the full phyllotactic rotation has completed before the next bifurcation.
Figure 4.11: A diagram showing the structure of *Horneophyton*’s aerial axes, including a simplified L-system interpretation. Sections of the axis are labelled as follows: I(\(w\)) internode segments, with width. \(+,-,(\theta)\) bends by angle \(\theta\). \(\{(\theta)\}\) roll by angle \(\theta\). A apices, with vigour and number of segments. S sporangium. \(t_t, t_b\) are the termination and branching thresholds, and \(x\) is the threshold number of internode segments before branching. \(\theta_B\) is the plant’s branching angle. \(\theta_D\) is the deflection angle. \(\theta_P\) is the phyllotactic divergence angle. \(f\) represents the ‘vigourDecay’ function.
phyllotactic angle, and deflected by $X \sim N(0, 5^2)^\circ$ to make the axes appear less rigid. Then, a new Internode module, representing one segment of the biological internode, is produced. It has the same age, length and vigour as its parent apex, and an initial width of 5 mm. If the Apex’s vigour is above the branching threshold and the fixed number segments has already been produced, two new Apex modules are produced to replace the old one, each on a separate branch deflected by the branching angle (Fig. 4.11).

Existing Internode modules do not produce any other modules in this model, but regulate their own age, length, and width in accordance with the pipe model. Like Apex modules, they age by a factor of $dt$ each time step and grow in length according to the ‘growthRate’ function. Any Internode with no other Internode following it maintains its original width. If there is another Internode following the currently considered Internode in the string, the current Internode module’s width is set to 1.01 times the width of its successor. Finally, an Internode that appears directly before a dichotomy has two successive Internode modules. Its width is set to $\sqrt{\text{width}_R^2 + \text{width}_L^2}$, where $\text{width}_R$ and $\text{width}_L$ are the widths of the Internode modules on the left and right branches (Fig. 4.11).

Because the sporangia of Horneophyton branch into multiple lobes, they are modelled similarly to the aerial axes, with sporangial Apex and Internode modules (Fig. 4.11). They begin as a Sporangium module, which is analogous to the Apex modules of the main axes. If
its vigour is greater than a fixed threshold, it can branch. Unlike the Apex, which branches when it reaches twelve segments, the sporangia branch with fewer segments depending on their vigour in order to give the sporangia their stubby, lobed shape. The number of segments generated before bifurcating is dependent on the vigour of the Sporangium. If the required number of segments is not yet met, a new segment can be generated. If the age of the Sporangium is greater than a fixed threshold the axis is rolled and deflected, and a SporangiumNode module is produced. A new contour is also selected to give the sporangium its irregular surface (Fig. 4.9). If none of these conditions is met, the Sporangium ages by $dt$, and grows in length by a scaled down version of ‘growthRate’. It terminates when it’s vigour drops below a small threshold. SporangiumNode modules simply grow in age and with in accordance with ‘growthRate’. Due to the theorized bag-like form of the sporangia, modelling them as a branching structure using generalized cylinders is not ideal (Fig. 4.12). Ideally the sporangia should be modelled as implicit surfaces based on branching skeletons, but this method is unavailable in L-studio.

During the interpretation step of the simulation, the modules described previously are translated into viewable generalized cylinders. Apex and Rhizome modules are rendered as a narrow cylinder tipped with a sphere, which terminate the axes when they are present. Internode, SporangiumNode and RhizomeInterior are rendered as cylinders of the prescribed width and length, and their colour is determined by the ‘colour’ function, based on their age. Sporangium modules are modelled to approximate the distinctive tip of the actual sporangia. Before rendering, the program checks whether or not the segments are within the view volume. If they are not, the rendering step is skipped to reduce processing time. The detail of the model is also determined by the camera’s distance from the Base module. If the camera is far away, the polygon count is reduced by decreasing the number of sides per generalized cylinder.

The code for the Horneophyton L-System is given in Appendix A.1.

---

6Divided by two, and inherently reduced due to the lower vigour of the sporangium.
4.3.4 Results

Final renderings of *Horneophyton* are presented in this section. Fig. 4.13 shows a mature aerial axis. Fig. 4.14 illustrates *Horneophyton*’s development over time. Fig. 4.15 shows details of *Horneophyton*’s axes and sporangia.

Figure 4.13: A single aerial axis of *Horneophyton* as it appears in the model.
Figure 4.14: *Horneophyton* at different stages during its development

Figure 4.15: A closer view of some of *Horneophyton*’s odd shaped sporangia, modelled within the limitations of L-systems. Right image edited to remove artefacts.
4.4 *Aglaophyton major*

4.4.1 Description

*Aglaophyton* is one of the better studied and more common plants of the Rhynie Chert. It was originally classified as *Rhynia gwynne-vaughanii* [40], and later as *Rhynia major* [42], due to its similarities to *Rhynia gwynne-vaughanii*, described in the next section. Significant structural differences were eventually found, and it was reclassified as *Aglaophyton major* [24]. Its aerial axes are similar to many other Rhynie plants, exhibiting repeated dichotomous branching and reaching a height of up to 15 cm [40]. Aerial axes reach a maximum diameter of 6 mm, and have been observed as small as 1 mm [40]. Axes dichotomize at an angle in the range of 60° to 90°.

The structure of its decumbent axes is of particular interest. They appear to be structurally similar to the aerial axes, but they bend over until they contact a suitable substrate [67]. Since nothing is known about how these plants actually grew, it is impossible to tell whether these axes grew vertically before bending over, or if they originally grew in a decumbent position, turning upward only locally (the latter assumption is used in this model).
When contact with the substrate is made, the axis anchors itself with unicellular rhizoids, and turns upright. This process repeats, with some axes remaining upright, and others turning downward (Fig. 4.17). As a result, the decumbent axes appear to ‘bounce’ along the ground. The diameter of these axes is similar to that of the aerial axes. They were also capable of branching dichotomously and repeatedly, allowing the plant to cover large areas of the substrate.

The sporangia of Aglaophyton were terminal, fusiform and relatively large, with an average length of 12 mm and a diameter of 4 mm. There seems to frequently be a dichotomy below the sporangia, causing them to appear in pairs [24]. These sporangia have been shown to dehisce lengthwise [66].

In situ fossils of Aglaophyton indicate that it preferred to grow on litter-covered, organic-
Figure 4.18: A. Longitudinal section of an *Aglaophyton* sporangium, terminally connected to an axis. B. A similar *Aglaophyton* sporangium as it appears in the model. C. A shrivelled, detached sporangium. D. A similar *Aglaophyton* sporangium as it appears in the model. Modified after plates from [24].
Figure 4.19: Selected previous reconstructions of *Aglaophyton*. Images from [24], [78] and [6] (with permission) respectively.

rich substrates rather than sinter. It is found both in monotypic stands, and in association with all other common Rhynie plants. The plant also seems to be drought resistant, due to adaptations that allow it to conserve water [58]. *Aglaophyton* is also associated with *in situ* algae, crustaceans, and fungi, however, suggesting that it was exposed to wet conditions during its life cycle, possibly during floods. Evidence suggests that *Aglaophyton* spores required wet conditions to germinate [67].

There are many reconstructions and artist’s renditions of *Aglaophyton*. A selection is presented in Fig. 4.19.
Table 4.5: Values and sources of important modelling parameters for *Aglaophyton*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Source</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width of aerial axes</td>
<td>&lt;6.0 mm</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Width of decumbent axes</td>
<td>&lt;6.0 mm</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Branching angle for aerial axes</td>
<td>(X \sim N(75, 8^2)^\circ)</td>
<td>24</td>
<td>Assumed to be the same as the aerial axes</td>
</tr>
<tr>
<td>Branching angle for decumbent axes</td>
<td>(X \sim N(75, 8^2)^\circ)</td>
<td>Assumed</td>
<td>Based on phyllotactic angle of 137.5(^\circ) observed in <em>Asteroxylon</em></td>
</tr>
<tr>
<td>Phyllotactic angle</td>
<td>(X \sim N(137.5, 5^2)^\circ)</td>
<td>Assumed</td>
<td></td>
</tr>
<tr>
<td>Length of internodes</td>
<td>Variable</td>
<td></td>
<td>Assumed</td>
</tr>
<tr>
<td>Height of plant</td>
<td>(~15\ cm)</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Dimensions of sporangia</td>
<td>12.0 mm x 4.0 mm</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

4.4.2 Modelling

The most important modelling parameters for *Aglaophyton* are listed in Table 4.5. The aerial axes grow for a normally distributed period of time representing one plastochron, and then bifurcate. Axes dichotomize at an angle in the range of \(X \sim N(75, 8^2)^\circ\) (Fig. 4.16). Based on their structure, it is assumed that the aerial axes of *Aglaophyton* exhibit a strong positive orthotropism, while the decumbent axes may show a negative gravitropism. Each internode segment is deflected upwards when it is produced, giving the axes their curved shape.

The behaviour of the apex is controlled by its individual vigour value. The plant may branch as long as the vigour is above the branching threshold, and once the vigour drops below that threshold, the axis has a chance to produce a sporangium or pair of sporangia during each time-step. If the axis fails to produce any sporangia before the vigour drops below the termination threshold, the axis terminates. When axes are first produced, they are 0.75 mm in width. As they mature, they thicken and usually reach a maximum diameter of up to 6.0 mm.

In the model, sporangia are produced on the ends of axes that have vigour within a certain threshold, terminating them. After growing to its full size of 12 mm by 4 mm, each
sporangium dehisces lengthwise. The character of the dehiscence was carefully captured in the way that the valves bend (Fig. 4.18).

The unique decumbent axes of *Aglaophyton* required a more complex modelling approach. The apex of the decumbent axis can exist in three states, depending on the stage of growth. The initial state is ‘Growing’ wherein axes that are not touching the ground grow, and slowly bend down over time. When the sensor for the axis detects a collision with the ground, the state is changed to ‘Reacting’, wherein the axis anchors itself and sharply turns upwards to avoid intersecting the ground. The sensor extends past the end of the axis in order to detect the ground before contact is made. This was done to reduce the complexity of the model. Ideally, axes would not react until direct contact was made, but this resulted in axes that intersected with the ground. The third state is ‘Rebounding’ wherein the axis continues to curve away from the ground, and can dichotomize to produce true aerial axes. The state then returns to ‘Growing’, and the process repeats as long as the axis’ vigour is greater than a small threshold.

4.4.3 Implementation

The process by which *Aglaophyton Germinator* modules are placed on the terrain is identical to *Horneophyton*, but the behaviour of the decumbent axis is entirely different.

The *Germinator* places a *RhizomeApex* module in addition to a *Base* module, which begins the process of producing a decumbent axis (Fig. 4.20). The *RhizomeApex* module grows each time step as long as its vigour remains above a fixed threshold. The ‘canBranch’ parameter of the apex (Table 4.6) is a boolean value that ensures that only one upright axis will emerge from each ‘arch’ of the decumbent axis. If ‘canBranch’ is true and the state of the apex is ‘Growing’, an aerial axis is produced. A new branch is created, which turns upwards by 30° and is tipped with an *Apex* module. All its parameters are zero except for vigour which is equal to the decumbent axis’ previous vigour plus a small, randomly
<table>
<thead>
<tr>
<th>Module</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germinator</td>
<td>delay</td>
<td>Places spore on the terrain, and tracks the delay before it germinates</td>
</tr>
<tr>
<td>RhizomeApex</td>
<td>age, length, vigour, state,</td>
<td>Determines when and how the decumbent axes grow, branch, turn upward, and</td>
</tr>
<tr>
<td></td>
<td>canBranch</td>
<td>terminate</td>
</tr>
<tr>
<td>Apex</td>
<td>age, length, vigour</td>
<td>Determines when and how the plant grows, branches, produces sporangia, and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>terminates</td>
</tr>
<tr>
<td>RhizomeInternode</td>
<td>age, length, width</td>
<td>Represents internal sections of the decumbent axes, and determines their</td>
</tr>
<tr>
<td></td>
<td></td>
<td>length and width</td>
</tr>
<tr>
<td>Internode</td>
<td>age, length, width</td>
<td>Represents internal sections of the aerial axes, and determines their length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and width</td>
</tr>
<tr>
<td>SporangiumGroup</td>
<td>(none)</td>
<td>Simple module that produces the sporangial stalk, and generates a single</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sporangium or a pair of sporangia</td>
</tr>
<tr>
<td>SporangiumStalk</td>
<td>segmentLength, numSegments</td>
<td>Represents the sporangial stalks of the plant</td>
</tr>
<tr>
<td>Sporangium</td>
<td>(none)</td>
<td>Simple module that places ‘HalfSporangium’ modules</td>
</tr>
<tr>
<td>HalfSporangium</td>
<td>age, length, bendFactor</td>
<td>Represents half of the body of a sporangium, to allow for dehiscence</td>
</tr>
<tr>
<td>Sensor</td>
<td>hit</td>
<td>Determines whether part of a decumbent axis has made contact with the terrain</td>
</tr>
<tr>
<td>Base</td>
<td>age, position</td>
<td>Stores the total age and position of the plant</td>
</tr>
</tbody>
</table>

Table 4.6: A list of the L-System modules used to model *Aglaophyton*

<table>
<thead>
<tr>
<th>Function</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>sporangiumWidth(n)</td>
<td>sporangiumWidthF(n) · factor</td>
</tr>
<tr>
<td>sporangiumSplit(n, age)</td>
<td>factor · sign ((a - \frac{n\cdot age}{b})) · (</td>
</tr>
</tbody>
</table>

Table 4.7: Functions used to model different aspects of *Aglaophyton*. sporangiumWidth(n) returns the width of the sporangium at each segment n, and sporangiumSplit(n, age) returns the angle between each segment n, which changes over time. See Fig. 4.22 for base functions. a, b, c, d, h, and factor are constants.
Figure 4.20: A diagram showing the structure of *Aglaophyton*’s decumbent axes, including a simplified L-system interpretation. Sections of the decumbent axis are labelled as follows: Ir decumbent axis interior segments. Ar(cb) decumbent axis apex, with ‘canBranch’ parameter. A aerial apex. I aerial internode. Red, orange and yellow apices signify ‘growing’, ‘reacting’ and ‘rebounding’ growth states respectively. The circles in the internodes show what the state was at that point, while a slash signifies that ‘canBranch’ is false. $\theta_A$ is the angle between decumbent and aerial axes. $\theta_G$ is the angle the axis bends downward during growth. $\theta_R$ is the angle the axis bends upward when reacting. $\theta_B$ is the angle the axis bends upward when rebounding. $\theta_t$ is the angle between the apex and the ground, and $t$ is a threshold of $\sim 45^\circ$. The dotted black line indicates ground level.

\[
\begin{align*}
\text{Ir} & \quad \text{Ar}, \text{Ar}, \text{Ar}(cb) \quad \text{A} \quad \text{I} \quad +, -, (\theta) \\
\text{Ar}(cb) & : \quad \text{cb} \rightarrow [+(\theta_A) \quad \text{A}] \quad \text{Ar} (\text{false}) \\
\text{Ar}(cb) & \rightarrow \text{Ir} \quad -(\theta_G) \quad \text{Ar}(cb) \\
\text{Ar}(cb) & : \quad \text{hitGround} \rightarrow +(\theta_R) \quad \text{Ir} \quad \text{Ar}(cb) \\
\text{Ar}(cb) & \rightarrow \text{Ir} \quad +(\theta_R) \quad \text{Ar} (\text{true}) \\
\text{Ar}(cb) & : \quad \theta_t < t \rightarrow +(\theta_b) \quad \text{Ir} \quad \text{Ar}(cb) \\
\text{Ar}(cb) & : \quad \theta_t \geq t \rightarrow +(\theta_b) \quad \text{Ir} \quad \text{Ar}(cb)
\end{align*}
\]
distributed amount. The elasticity of the axis is set to a positive value to give the aerial axis its upwards tropism. Whether or not an aerial axis is formed, further productions are applied to the RhizomeApex. If the apex’s state is ‘Reacting’, then the axis bends upwards by $0^\circ$ to $\sim 52^\circ$ depending on the axis’ current orientation and a random distribution. After the axis reacts, a new ‘arch’ has been started and ‘canBranch’ is set to true. As long as the module’s vigour remains above a larger threshold, it has a chance of branching each time step after its length reaches nine segments. If this happens, two new RhizomeApex modules are produced with the same vigour value as their parent, deflected by the decumbent axis branching angle. Otherwise, if the apex’s age has reached fixed value, a RhizomeInternode module is produced, which is deflected upwards or downwards depending on its current state. If the state is ‘Growing’, the deflection is $0^\circ$ to $\sim 56^\circ$ downward depending on its current orientation. If the state is ‘Rebounding’, the axis bends upward by $0^\circ$ to $\sim 21^\circ$ depending on its current orientation. In the ‘Rebounding’ state, the axis reverts to ‘Growing’ if its upward angle reaches $\sim 45^\circ$. In any state, the axis is also deflected horizontally by $X \sim N(0, 6^2) ^\circ$ to make the axis more realistic. Finally, if no special conditions are met, the RhizomeApex ages by $dt$, increases in length according to the ‘growthRate’ function, and loses vigour according to the ‘vigourDecay’ function (Table 4.4).

RhizomeInternode modules of Aglaophyton are managed in the same way as the Internode modules in most of these models. Their internal parameters change over time, but they do not produce any new structural modules. The pipe model is used to dictate their width, using an exponent of five. RhizomeInternode modules also slowly settle onto the ground over time if they are slightly elevated. When they make contact, a module named Stop is created, which stops the axes from bending further when it is in their right context.

When the RhizomeApex produces aerial axes, the Apex module takes over. Apex modules of Aglaophyton are managed in the same way as the Apex modules in most of these models, although a pipe model exponent of three was used to achieve the desired diameter of
Aglaophyton’s axes, though it differs from some in how it produces sporangia. While the vigour is between the branching and termination thresholds, there is a 15% chance that a SporangiumGroup module will be produced each timestep and terminate the axis. If no SporangiumGroup has been produced before the vigour drops below the termination threshold, the axis terminates with no sporangia. The ‘Internode’ modules produced by the ‘Apex’ are almost identical to the RhizomeInternode modules, except that they lack sensors.

The SporangiumGroup modules are a placeholder, which immediately generate a sporangial stalk and one or two Sporangium modules (Fig. 4.21). The Sporangium modules then generate two HalfSporangium modules on separate branches, and a number of internal spores. The HalfSporangium modules utilize a semicircular contour to give them their shape. The longitudinal contour of the sporangia is determined by the ‘sporangiumWidth’ function (Table 4.7). Depending on their age, a bend is applied to each HalfSporangium, causing them to peel apart, then curve back inward, to represent their dehiscence.

During the interpretation step of the simulation, the modules described previously are translated into viewable generalized cylinders. Apex and Rhizome are rendered as a narrow cylinder tipped with a sphere, to terminate the axis if they are present. Internode and RhizomeInternode modules are rendered as cylinders of the prescribed width and length, and their colour is determined by the ‘colour’ function, based on their age. Before rendering, the program checks whether or not the segments are within the view volume. If they are not, the rendering step is skipped to reduce processing time. The detail of the model is also determined by the camera’s distance from the Base module. If the camera is far away, the polygon count is reduced by decreasing the number of sides per generalized cylinder. From a distance, HalfSporangium modules are also modelled with fewer segments lengthwise to reduce the polygon count. Spores inside dehisced sporangia are also rendered in reduced detail from a distance.

The code for the Aglaophyton L-System is given in Appendix A.2.
Figure 4.21: A diagram showing the structure of *Aglaophyton*’s sporangia, including a simplified L-system interpretation. Sections of the structure are labelled as follows: I aerial internode. Sg sporangium group. S sporangium placeholder. Sh half sporangium. Ss sporangium stalk. +,-,(θ) bends by angle θ. \(\theta\) roll by angle θ. \(\theta_B\) is the branching angle between sporangial stalks. \(P_1\) and \(P_2\) are the probabilities of forming a pair of sporangia or a single sporangium, respectively. \(f\) is a function dictating the bend between each half sporangium segment over time. \(Ss \rightarrow F F F\) is used because a single segment cannot exhibit orthotropism.
Figure 4.22: The base functions used to define different aspects of Aglaophyton’s form. 

**A. sporangiumWidthF**, an approximation of the longitudinal section of Aglaophyton’s sporangia. 

**B. sporangiumSplitF**, causes the sporangia to rapidly but smoothly split open after some time.

### 4.4.4 Results

Final renderings of Aglaophyton are presented in this section. Fig. 4.23 shows a mature plant. Fig. 4.24 shows details of Aglaophyton’s axes and sporangia. Fig. 4.25 illustrates Aglaophyton’s development over time.
Figure 4.23: A single model of an *Aglaoophyton* plant with multiple aerial axes arising from a system of decumbent axes.

Figure 4.24: Details of *Aglaoophyton*’s decumbent axes and dehiscent sporangia.
Figure 4.25: *Aglaophyton* at different stages during its development.
4.5  *Rhynia gwynne-vaughanii*

4.5.1 Description

*Rhynia* is probably the most abundant plant in the Rhynie chert, as well as the first to be discovered. Its aerial axes are somewhat more complex than those of other Rhynie plants, exhibiting repeated dichotomous branching, as well as adventitious or monopodial branches, which appear to be the fertile regions of the plant (Fig. 4.26). All together, the aerial axes could reach a height of up to 20 cm [40]. The cylindrical parts of the axes reach a maximum diameter of 3 mm, although some parts of the surface bear small bumps. Axes dichotomize at an angle in the range of $17^\circ$ to $35^\circ$ [22].

The decumbent axes of *Rhynia* are structurally similar to the aerial axes, but they lie directly on the substrate, turning locally upward to form the aerial axes [22]. The bumps on the decumbent axes bear tufts of unicellular rhizoids that anchor it to the ground. Occasionally, the bumps of the aerial axes may also bear rhizoids, similar to the adventitious buds of extant plants. The diameter of these axes is similar to that of the aerial axes, and they appear to branch at similar intervals and angles.

The sporangia of *Rhynia* were terminal and fusiform in shape, with an average length of 3.6 mm and a diameter of 2.4 mm. The sporangia always appear on the tips of adventitious branches of fertile aerial axes. No dehiscence mechanism has been observed, but a layer of cells at the base of the sporangium is interpreted as the point of abscission [22].

Fossil specimens of *Rhynia* are relatively common, likely due to the number of Rhynia plants in the ecosystem, and the amount of ground they covered due to their spreading decumbent axes. Specimens of *Rhynia* found in growth position are often found isolated from other Rhynie plants, suggesting that it grew in monotypic stands, and was therefore an early colonizer of sinter substrates [83]. However, *Rhynia* is found in association with all other Rhynie plants in other deposits, suggesting that it was highly tolerant to a variety of conditions and could withstand competition [58].
Figure 4.26: Reconstructed fossil axes of *Rhynia*, compared to similar axes from the model. Modified after plates from [22].

Figure 4.27: Selected previous reconstructions of *Rhynia*. Images from [22], [4] and [31] respectively.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Source</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width of aerial axes</td>
<td>&lt;3.0 mm</td>
<td>[40]</td>
<td></td>
</tr>
<tr>
<td>Width of decumbent axes</td>
<td>&lt;3.0 mm</td>
<td>[40]</td>
<td></td>
</tr>
<tr>
<td>Branching angle for aerial axes</td>
<td>(X \sim N(26,5^2))°</td>
<td>[22]</td>
<td>Assumed</td>
</tr>
<tr>
<td>Branching angle for decumbent axes</td>
<td>(X \sim N(26,5^2))°</td>
<td>Assumed</td>
<td>Assumed to be the same as the aerial axes</td>
</tr>
<tr>
<td>Phyllotactic angle</td>
<td>(X \sim N(137.5,5^2))°</td>
<td>Assumed</td>
<td>Based on phyllotactic angle of 137.5° observed in <em>Asteroxylon</em></td>
</tr>
<tr>
<td>Length of internodes</td>
<td>Variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Height of plant</td>
<td>(~20) cm</td>
<td>[40]</td>
<td></td>
</tr>
<tr>
<td>Dimensions of sporangia</td>
<td>3.6 mm x 2.4 mm</td>
<td>[40]</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.8: Values and sources of important modelling parameters for *Rhynia*

There are many reconstructions and artists renditions of *Rhynia*. A selection is presented in Fig. 4.27.

4.5.2 Modelling

The most important modelling parameters of Rhynia are listed in table 4.8. The aerial axes grow for a normally distributed period of time representing one plastochron, and then bifurcate. Axes dichotomize at an angle in the range of \(X \sim N(26,5^2)\)°. Based on their structure, it is assumed that the aerial axes of *Rhynia* exhibit a weak positive orthotropism, while the decumbent axes directly follow the substrate. Each aerial internode segment is deflected slightly upwards when it is produced, giving the aerial axes their curved shape.

The behaviour of the apex is controlled by its individual vigour value. The plant may branch dichotomously as long as the vigour is above the branching threshold. Fertile adventitious axes can emerge from internodes during later time-steps. Once the vigour drops below the branching threshold, the axis has a chance to produce a sporangium during each time-step if the axis is fertile. If the axis fails to produce any sporangia before the vigour drops below the termination threshold, or if the axis was not a fertile adventitious axis, it terminates. When axes are first produced, they are 0.5 mm in width. As they mature, they thicken and usually reach a maximum diameter of up to 3.0 mm.
Sporangia are produced on the ends of adventitious axes that have vigour within a certain range, terminating them. *Rhynia*’s sporangia do not dehisce, so they are modelled as a single surface.

The decumbent axes of *Rhynia* form a branching network on the surface of the substrate, which intermittently gives rise to aerial axes. These axes grow and branch in a similar fashion to the aerial axes, but sensors in the apex keep them close to the surface of the substrate. The axes will bend upwards whenever they contact the surface, and generally bend downwards otherwise, keeping them in contact with the surface until an aerial axis is formed.
Figure 4.28: A diagram showing the structure of Rhynia’s decumbent axes, including a simplified L-system interpretation. Sections of the decumbent axis are labelled as follows: Ir decumbent axis interior segments. Ar(lb) decumbent axis apex, with ‘lastBranch’ parameter. A aerial apex. I aerial internode. θA is the angle between decumbent and aerial axes. θC is the angle the axis bends downward during growth. θR is the angle the axis bends upward when reacting. t is a threshold of that determines when an aerial axis will be produced. The dotted black line indicates ground level.

4.5.3 Implementation

The simulation begins with an axiom of one or more Germinator modules. Each Germinator has its initial delay decremented by one until it reaches zero, at which point it produces a Rhynia Base and RhizomeApex module at a random location on the terrain surface. The position of the plant is stored as a parameter of Base. Meanwhile, the RhizomeApex module is initialized with a random vigour value, and all other parameters zero. The RhizomeApex module grows each time step as long as its vigour remains above its termination threshold (Fig. 4.28). The ‘lastBranch’ parameter of the apex (Table 4.9) tracks the distance between successive aerial axes emerging from the decumbent axis. If ‘lastBranch’ is greater than a fixed threshold, an aerial axis is produced. A new branch is created, which turns upwards by $\leq 75^\circ$, depending on its orientation⁷ and is tipped with an Apex module, with all its parameters set to zero except for vigour which is equal to the decumbent axis’ previous

\[ \text{Ir Ar(lb) A I } +, -(\theta) \]

\[ \text{Ar(lb)} : \text{lb} > t \rightarrow [+(\theta_A) \ A] \ Ar(0) \]

\[ \text{Ar(lb)} : \text{hitGround} \rightarrow +(\theta_R) \ Ir \ Ar(lb++) \]

\[ \text{Ar(lb)} \rightarrow \text{Ir} - (\theta_A) \ Ar(lb++) \]

⁷The angle at which the axis turns upwards is multiplied by $1 - yh$, where $yh$ is the y component of the apex’s heading.
vigour plus a small, randomly distributed amount. The elasticity of the axis is set to a very small value to give the aerial axis its minor upwards tropism. Whether or not an aerial axis is formed, further productions are applied to the RhizomeApex. As long as the module’s vigour remains above its branching threshold, it has a chance of branching during each time step after its length reaches eighteen segments. If this happens, two new RhizomeApex modules are produced with the same vigour and ‘lastBranch’ values as their parent, deflected by the decumbent axis branching angle. Otherwise, if the apex’s age has reached a fixed threshold, a RhizomeInternode module is produced, which is deflected upwards or downwards depending on its location to keep it relatively horizontal, but on the surface of the substrate. It is also deflected horizontally by \( X \sim N(0, 4^2) \degree \) to give it a less rigid appearance. Finally, if no special conditions are met, the RhizomeApex ages by \( dt \), increases in length according to the ‘growthRate’ function, loses vigour according to the ‘vigourDecay’ function, and ‘lastBranch’ is incremented. Once RhizomeInternode modules have been produced, they are handled the in the same way as Aglaophyton’s RhizomeInternode modules.

When the RhizomeApex produces aerial axes, the Apex module takes over. Apex modules of Rhynia are managed in the same way as the Apex modules in most of these models, but some aspects are handled slightly differently. While the vigour is between the branching and termination thresholds, and if the axis is ‘fertile’, a Sporangium module is produced, terminating the axis (Table 4.9). If no Sporangium has been produced before the vigour drops below the termination threshold, the axis terminates regardless. The Internode modules produced by the Apex are similar to the RhizomeInternode modules, except that they lack sensors, and they have a small chance of producing a fertile, adventitious axis.

The interpretation step of Rhynia is identical to that of Aglaophyton, except for the interpretation of the sporangia. The sporangia of Rhynia are modelled as simple surfaces, since they lack distinctive features.

The code for the Rhynia L-System is given in Appendix A.3.
4.5.4 Results

Final renderings of *Rhynia* are presented in this section. Fig. 4.29 illustrates *Rhynia*’s development over time. Fig. 4.30 shows a mature plant. Fig. 4.31 shows details of *Rhynia*’s axes and sporangia.

Figure 4.29: *Rhynia* at different stages during its development.
Figure 4.30: A single model of a *Rhynia* plant with multiple aerial axes arising from a system of decumbent axes.

Figure 4.31: A group of aerial axes of *Rhynia* bearing sporangia.
Figure 4.32: A fossilized aerial axis of *Nothia*, compared to its counterpart in the model. Fossil image taken from [39]

4.6 *Nothia aphylla*

4.6.1 Description

*Nothia* is the another commonly found fossil plant in the Rhynie area, although less common than *Horneophyton*. Specimens of *Nothia* were originally thought to be part of *Asteroxylon* [42], but were later determined to be part of a separate species. Its most distinctive feature is its irregular cortex. Similar to many plants in the Rhynie ecosystem, the primary structure of *Nothia* is composed of repeatedly dichotomized axes, which reach a maximum height of approximately 15 cm. The maximum width of the axes was about 2.5 mm to 5.0 mm [27].

Multiple aerial axes were likely connected by a shallow subterranean rhizome system. The rhizomatous axes have never been found in anatomical connection with complete aerial axes, but they intermittently turn upright, likely representing the transition to aerial axes [39]. As much as the lower 8 mm of the aerial axes have been preserved in connection with
the rhizomatous axes, suggesting these were buried as deep as 8 mm in the substrate. Further sections of the rhizome branch from the lower 3 mm of the upright axes at near right angles.

The sporangia of *Nothia* were reniform, with maximum dimensions of approximately 3.1 x 1.8 x 1.4 mm. They were borne both laterally and terminally. These sporangia appear in many different arrangements, including pairs, columnar whorls, terminal clusters, and apparently random arrangements (Fig. 4.33). The non-terminal sporangia branch off at acute angles from the main axis [27]. They have been shown to dehisce laterally (Fig. 4.34).

*Nothia*’s rhizomatous axes are usually much better preserved than its aerial axes, suggesting that the aerial axes were short lived, perhaps seasonal [39]. Information on the palaeoecology of *Nothia* is scarce because the aerial axes are rarely found in life position. Rhizome axes are occasionally found in monotypic assemblages, or penetrating litter of other plants, commonly *Asteroxylon*. This suggests that *Nothia* grew in sandy litter substrates, and preferred similar environments to *Asteroxylon* [79].

Several previous reconstructions of *Nothia* are available, and are shown in Figure 4.35. These reconstructions generally appear realistic and are accurate to current data, but are static.

### 4.6.2 Modelling

The most important modelling parameters of Nothia are listed in table 4.10. Each aerial axis grows for a normally distributed period of time representing one plastochron, and then bifurcates. The angle of dichotomy for *Nothia* is relatively wide, at $X \sim N(80, 10^2)\,^\circ$ (Fig. 4.32). Based on the structure of its aerial axes from fossil evidence, it is assumed that *Nothia* is strongly orthotropic. Each internode segment is deflected upwards when it is produced, giving the axes their curved shape.

The rate at which each aerial axis grows and the time at which its development terminates are controlled by the vigour of that axis’ apex. The plant may branch as long as the vigour is above the branching threshold, and sporangia are produced in different arrangements while
Figure 4.33: Previous reconstructions of *Nothia* sporangial arrangements (left), modified after diagrams from [27], compared with the model (right). A. B. Columns of sporangia. C. Sporangia arranged in a whorl. D. A terminal cluster.

Figure 4.34:  A A fossilized sporangium of *Nothia*, showing lateral dehiscence, modified after plates from [27]. B. A sporangium from the model.
Figure 4.35: Selected previous reconstructions of *Nothia*. Images from [27], [39] and [82] respectively

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Source</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width of aerial axes</td>
<td>&lt;2.5 mm</td>
<td>[82]</td>
<td></td>
</tr>
<tr>
<td>Width of rhizomatous axes</td>
<td>&lt;2.3 mm</td>
<td>[39]</td>
<td></td>
</tr>
<tr>
<td>Branching angle for aerial axes</td>
<td>$X \sim N(80, 10^2) ^\circ$</td>
<td>[27]</td>
<td></td>
</tr>
<tr>
<td>Branching angle for rhizomatous axes</td>
<td>$X \sim N(52, 16^2) ^\circ$</td>
<td>Assumed</td>
<td>Simplified from actual rhizome structure</td>
</tr>
<tr>
<td>Phyllotactic angle</td>
<td>$X \sim N(137.5, 5^2)^\circ$</td>
<td>Assumed</td>
<td>Based on phyllotactic angle of 137.5° observed in <em>Asteroxylon</em></td>
</tr>
<tr>
<td>Length of internodes</td>
<td>Variable</td>
<td>Assumed</td>
<td></td>
</tr>
<tr>
<td>Height of plant</td>
<td>$\sim$15 cm</td>
<td>[27]</td>
<td></td>
</tr>
<tr>
<td>Dimensions of sporangia</td>
<td>3.1 mm x 1.8 mm x 1.4 mm</td>
<td>[27]</td>
<td></td>
</tr>
<tr>
<td>Angle of sporangia</td>
<td>$X \sim N(30, 5^2)^\circ$</td>
<td>[27]</td>
<td>Described as ‘acute angle’, with a 30° dichotomy pictured.</td>
</tr>
</tbody>
</table>

Table 4.10: Values and sources of important modelling parameters for *Nothia*
the vigour is between the thresholds. The axis terminates with a terminal sporangium group when the vigour drops below the termination threshold. When axes are first produced, they are 0.75 mm in width. As they mature, they thicken and usually reach a maximum diameter of about 2.5 mm. The axes are nearly cylindrical near their bases, but further up the axes the cortex becomes more irregular.

*Nothia*’s sporangia begin to appear in columns along the terminal axes once the vigour of the associated apex drops below a fixed threshold. The sporangia were modelled in two halves to allow them to dehisce once they reach a fixed age. The non-terminal sporangia branch off the parent axis at an angle of \( X \sim N(35, 5^2) \)°.

The rhizomatous axes of *Nothia* form a branching subterranean network which intermittently gives rise to aerial axes. The rhizomatous axes grow and branch in a similar fashion to the aerial axes, but sensors in the apices keep them close beneath the surface of the substrate. The axes will bend downwards whenever they grow too close to the surface, and generally bend upwards otherwise, keeping them below the surface until an aerial axis is formed. The axes in the model are largely simplified compared to those of the actual plant, since they are not visible and serve mainly to determine the positions of the aerial axes.

### 4.6.3 Implementation

*Nothia*’s development begins in much the same way as *Rhynia*’s does. The **Germinator** produces a **Base** module and a **RhizomeApex** module at a random location below the terrain surface, beginning the plant’s growth. However, because *Nothia*’s rhizomatous axes are subterranean, they were modelled with minimal detail and serve mainly to distribute the aerial axes logically. The subterranean axis grows horizontally, and is deflected upwards or downwards depending on the position of the terrain to keep it below the surface.

The aerial axes of *Nothia* are generated in much the same way as the other Rhynie plants, with slight variations in branching angle and tropism to give them their unique character (Table 4.11). This model differs in the placement of its sporangia and the appearance
<table>
<thead>
<tr>
<th>Module</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germinator</td>
<td>delay</td>
<td>Places spore on the terrain, and tracks the delay before it germinates</td>
</tr>
<tr>
<td>RhizomeApex</td>
<td>age, length, vigour, last-Branch</td>
<td>Determines when and how the rhizomatus axes grow, branch, turn upward, and terminate</td>
</tr>
<tr>
<td>Apex</td>
<td>age, length, vigour</td>
<td>Determines when and how the plant grows, branches, produces sporangia, and terminates</td>
</tr>
<tr>
<td>RhizomeInternode</td>
<td>age, length, width</td>
<td>Represents internal sections of the rhizomatus axes, and determines their length and width</td>
</tr>
<tr>
<td>Internode</td>
<td>age, length, width</td>
<td>Represents internal sections of the aerial axes, and determines their length and width</td>
</tr>
<tr>
<td>Sporangium</td>
<td>(none)</td>
<td>Simple module that produces the sporangial stalk and places ‘HalfSporangium’ modules</td>
</tr>
<tr>
<td>HalfSporangium</td>
<td>age, length</td>
<td>Represents half of the body of a sporangium, to allow for dehiscence</td>
</tr>
<tr>
<td>Sensor</td>
<td>hit</td>
<td>Determines whether part of a rhizomatus axis has made contact with the terrain surface</td>
</tr>
<tr>
<td>Base</td>
<td>age, position</td>
<td>Stores the total age and position of the plant</td>
</tr>
</tbody>
</table>

Table 4.11: A list of the L-System modules used to model *Nothia*

---

Figure 4.36: The contour lines used to give the sporangia and axes of *Nothia* their shape
Figure 4.37: The base functions used to define different aspects of Nothia’s form. A. sporangiumWidthF, an approximation of the longitudinal section of Nothia’s sporangia. B. sporangiumSplitF, causes the sporangia to rapidly but smoothly split open after some time. of the internodes. The sporangia can be placed in columns, whorls, terminal clusters, or randomly. With the exception of terminal clusters, the type of sporangial arrangement is chosen randomly due to lack of evidence. Internode modules require more attention because of their unique shape. A linear blend between the standard circular contour and some more irregular contours (Fig. 4.36) along the length of the cylinder gives them their distinctive rough surface.

The Sporangium modules are a placeholder, which immediately generate a sporangial stalk and two HalfSporangium modules on separate branches. The HalfSporangium modules utilize a semicircular contour (Fig. 4.36) to give them their shape. The longitudinal contour of the sporangia is determined by the ‘sporangiumWidth’ function (Fig. 4.37). Over time the two HalfSporangium modules split apart to simulate dehiscence.

During the interpretation step of the simulation, the modules described previously are translated into generalized cylinders. Apex and Rhizome are rendered as narrow cylinders tipped with spheres, which terminate the axes when they are present. RhizomeInternode modules are rendered as cylinders of the prescribed width and length, and their colour is
determined by the ‘colour’ function, based on their age. Before rendering, the program checks whether or not the segments are within the view volume. If they are not, the rendering step is skipped to reduce processing time. The detail of the model is also determined by the camera’s distance from the Base module. If the camera is far away, the polygon count is reduced by decreasing the number of sides per generalized cylinder. From a distance, HalfSporangium modules are also modelled with fewer segments lengthwise due to their higher level of detail. Internode modules have the detail of their surface reduced as well.

The code for the Nothia L-System is given in Appendix A.4

4.6.4 Results

Final renderings of Nothia are presented in this section. Fig. 4.38 shows a mature plant. Fig. 4.39 shows details of Nothia’s axes and sporangia. Fig. 4.40 illustrates Nothia’s development over time.
Figure 4.38: A group of aerial axes of *Nothia*, as they appear in the model. The group is connected together by subterranean rhizomatous axes.

Figure 4.39: A close-up of *Nothia*’s irregular surface and sporangia.
Figure 4.40: *Notthia* at different stages during its development.
4.7 Ventarura lyonii

4.7.1 Description

Ventarura is less known, and less abundant than many other plants in this ecosystem. It is structurally similar to Nothia, but differs in the organization of its epidermis and cortex, and it has thicker aerial axes, ranging from 3.6 mm to 7.2 mm in diameter [57]. These axes also appear to branch at very small angles, between 25° and 45° [57]. The maximum recorded length of an aerial axis is 12 cm, but no complete axes have been discovered yet, and thus the maximum height is unknown. Ventarura fossils are often discovered in allochthonous deposits, thus its preservation is often poor. The fragmentary preservation of its axes also makes it difficult to tell where and how frequently the axes branched. Current evidence suggests that most axes branch infrequently, but others branch repeatedly within a short span [57]. Little is known about the rhizomatous axes of Ventarura, as they have not been found in anatomical connection with the aerial axes.

The sporangia of Ventarura were borne in strobili, with each sporangium attached perpendicular to its parent axis (Fig. 4.41A). Columns of sporangia appear vertically above one another, suggesting that the phyllotactic angle of this plant was a factor of 360° rather than 137.5°. The sporangia were formed in two halves, connected along their margins, indicating that these sporangia dehisced transversely (Fig. 4.41B). The average width of discovered sporangia is 3.5 mm long by 1.9 mm thick by 4.4 mm wide.

Because Ventarura fossils are often found transported by water, and not in life position, it is difficult to make conclusions about their original habitat. However, this makes it seem likely that Ventarura lived on or near the edges of bodies of water, which carried them to their deposition location. Due to the presence of apparently in situ rhizomatous axes, it is assumed that Ventarura preferred organic-rich and sandy substrates [85].

There exists only one preliminary reconstruction of Ventarura, based on the limited evidence available (Fig. 4.42).
Figure 4.41: A. A column of fossil *Ventarura* sporangia, from [57]. B. A single sporangium fossil, showing the line of dehiscence, from [85]. C. A column of sporangia from the model for comparison.

Figure 4.42: Preliminary reconstruction of *Ventarura*, modified after [85].
Table 4.12: Values and sources of important modelling parameters for *Ventarura*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Source</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width of aerial axes</td>
<td>3.6 mm - 7.2 mm</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>Branching angle for aerial</td>
<td>$X \sim N(36,6^2),^\circ$</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>axes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phyllotactic angle</td>
<td>$X \sim N(90,3^2),^\circ$</td>
<td>Assumed</td>
<td>Based on the arrangement of the sporangia, and the spacing between</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the sporangia that results from this choice.</td>
</tr>
<tr>
<td>Length of internodes</td>
<td>Variable</td>
<td>Assumed</td>
<td></td>
</tr>
<tr>
<td>Height of plant</td>
<td>10 cm - 16 cm</td>
<td>Assumed</td>
<td>Based on 12 cm partial axis described in 57</td>
</tr>
<tr>
<td>Dimensions of sporangia</td>
<td>3.5 mm x 1.9 mm x 4.4 mm</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>Angle of sporangia</td>
<td>$X \sim N(90,9^2),^\circ$</td>
<td>57</td>
<td>Described as perpendicular to axis.</td>
</tr>
</tbody>
</table>

4.7.2 Modelling

The most important modelling parameters of *Ventarura* are listed in table 4.12. *Ventarura’s* overall structure is very similar to that of *Horneophyton* and *Nothia*, and as such there are only minor differences in the model to reflect *Ventarura’s* unique features. Each axis grows for a normally distributed period of time representing one plastochron, and then bifurcates. Nearer to the top of the plant, the period of time between dichotomies is reduced, to represent the parts of the plant where branching points are more frequent. *Ventarura’s* aerial axes dichotomized at a relatively narrow angle, which is modelled as $X \sim N(35,6^2)\,^\circ$ to cover the observed range of 25° and 45°. Like *Horneophyton* and *Nothia*, *Ventarura* appears to be orthotropic. Each internode segment is deflected upwards when it is produced, giving the axes their curved shape. Since the maximum height of the plant was at least 12 cm, likely more [85], the model has been written to produce specimens between 10 cm and 16 cm tall.

The behaviour of the apex is controlled by its individual vigour value. The plant may branch as long as the vigour is above the branching threshold, and columns of sporangia are produced while the vigour is between the branching and termination thresholds. The axis terminates when the vigour drops below the termination threshold. When axes are first
Module Parameters Description
---
Germinator delay Places spore on the terrain, and tracks the delay before it germinates
Apex age, length, vigour, fertile Determines when and how the plant grows, branches, produces sporangia, and terminates
Internode age, length, width Represents internal sections of the aerial axes, and determines their length and width
Sporangium (none) Simple module that produces the sporangial stalk and places ‘HalfSporangium’ modules
HalfSporangium age, length Represents half of the body of a sporangium, to allow for dehiscence
Base age, position Stores the total age and position of the plant

Table 4.13: A list of the L-System modules used to model *Ventarura*

produced, they are 0.75 mm in width. As they mature, they thicken and reach a maximum diameter of about 3.6 to 7.2 mm.

Axes bearing sporangia are produced by the apex while its vigour is within a fixed range, after which the axis will terminate. The phyllotactic angle of the plant is assumed to be 90° due to the arrangement of the sporangia, and the spacing that results (Fig. 4.41A). The sporangia were modelled in two halves to allow them to dehisce once they reach a fixed age. The sporangia branch off the parent axis at an angle of $X \sim N(90, 6^2)°$.

The rhizomatous axes were not modelled due to lack of information.

### 4.7.3 Implementation

The model of *Ventarura* is implemented in a very similar manner to the previous models, especially *Nothia*. Since the rhizomes were not modelled, the Germinator produces a Ventarura Base and Apex module at a random location on the terrain surface once it’s ‘delay’ expires. The growth of the aerial axes is then almost identical, with a few differences. The phyllotactic divergence angle of *Ventarura* is assumed to be 90°, and so each internode segment is rolled by one tenth of this angle, and branches are produced after ten internode segments. *Ventarura*’s sporangia are produced on terminal axes that are between the branching and termination thresholds as usual. However, the angle between each successive sporangia is
Figure 4.43: The width function and cylinder contour used to give Ventarura’s sporangia their shape.

90°, and columns of sporangia only appear on axes with a apex flagged as ‘fertile’ (Table 4.13). Ventarura’s sporangia branch from their parent axes at an angle of $X \sim N(90, 5^2)$°.

The internodes of Ventarura are implemented in the same way as the internodes of the other Rhynie plants, but are generated with a greater initial width, such that they achieve the desired final width.

The Sporangium modules are a placeholder, which immediately generate a sporangial stalk and two HalfSporangium modules on separate branches. The HalfSporangium modules utilize a semi-ellipsoid contour (Fig. 4.43) to give them their shape. The longitudinal contour of the sporangia is determined by the ‘sporangiumWidth’ function (Fig. 4.43). Depending on their age, a bend is applied to each HalfSporangium, away from the other half, causing them to dehisce.

During the interpretation step of the simulation, the modules described previously are translated into generalized cylinders. This is achieved in the same manner as with the other Rhynie plants.

The code for the Ventarura L-System is given in Appendix A.5.
4.7.4 Results

Final renderings of *Ventarura* are presented in this section. Fig. 4.44 illustrates *Ventarura*’s development over time. Fig. 4.45 shows a mature plant. Fig. 4.46 shows details of *Ventarura*’s axes and sporangia.

Figure 4.44: *Ventarura* at different stages during its development.
Figure 4.45: A single aerial axis of *Ventarura* as it appears in the model.

Figure 4.46: A close-up of an aerial axis of *Ventarura*, bearing dehisced sporangia.
4.8 *Trichopherophyton teuchansii*

4.8.1 Description

*Trichopherophyton* is the most recently discovered and the scarcest of the Rhynie plants. It exhibits pseudomonopodial and dichotomous branching at an average angle of \(25^\circ\) [49]. *Trichopherophyton* is unique in that it bears tiny spinose projections on its epidermis, which range in length from 200\(\mu m\) to 400\(\mu m\), and sometimes reach even 1 mm (Fig. 4.47). It is the only plant of the Rhynie Chert that exhibits circinate vernation (Fig. 4.48). The aerial axes have a maximum diameter of 2.5 mm. The average height of the plant is unknown, so it is assumed to be similar to the other plants of this ecosystem, with an average height of 12 cm to 15 cm. Rhizomes have never been found in anatomical connection with the aerial axes, so little is known about them. Some fossils have been assumed to represent the rhizomatous axes of *Trichopherophyton* [84].

Little is known about the sporangia of *Trichopherophyton*, but they appear to be reniform with a lateral dehiscence mechanism. They have cross-sectional dimensions of up to 3.7 mm by 2.5 mm. They were connected to fertile axes laterally, but their arrangement is unknown.

Although specimens of *Trichopherophyton* are rare and poorly preserved, their locations are telling of its possible palaeoecology. It is plausible that *Trichopherophyton* was rare within the ecosystem, but there could be a bias in the fossils that were preserved or discovered. Fossils are found both in growth position and in plant litter, and they are always associated with other plants, such as *Nothia*, *Horneophyton* and *Rhynia*, suggesting that *Trichopherophyton* was a late colonizer of humic substrates [84].

4.8.2 Modelling

No reconstruction of this plant has previously been attempted. Due to lack of evidence, this is a highly speculative model, and parts of its reconstruction were based on related, better known plants. *Trichopherophyton* is part of the class *Zosterophyllopsida*, and appears to
Figure 4.47: A. Longitudinal section showing the spinose projections of *Trichopherophyton* [84]. B. An axis with spines as it appears in the model.

Figure 4.48: A. Longitudinal section showing a circinate apex of *Trichopherophyton* [49]. B. A partially coiled apex as it appears in the model.
Table 4.14: Values and sources of important modelling parameters for *Trichopherophyton*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Source</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width of aerial axes</td>
<td>&lt;2.5 mm</td>
<td>[49]</td>
<td></td>
</tr>
<tr>
<td>Branching angle for aerial</td>
<td>$X \sim N(25, 6^2)^\circ$</td>
<td>[49]</td>
<td></td>
</tr>
<tr>
<td>axes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phyllotactic angle</td>
<td>$X \sim N(137.5, 5^2)^\circ$</td>
<td>Assumed</td>
<td>Based on phyllotactic angle of 137.5° observed in <em>Asteroxylon</em>.</td>
</tr>
<tr>
<td>Length of internodes</td>
<td>Variable</td>
<td>Assumed</td>
<td></td>
</tr>
<tr>
<td>Height of plant</td>
<td>12 cm - 15 cm</td>
<td>Assumed</td>
<td></td>
</tr>
<tr>
<td>Dimensions of sporangia</td>
<td>3.7 mm x 2.5 mm</td>
<td>[49]</td>
<td></td>
</tr>
<tr>
<td>Angle of sporangia</td>
<td>$X \sim N(90, 9^2)^\circ$</td>
<td>[49]</td>
<td>Described as perpendicular to axis.</td>
</tr>
</tbody>
</table>

*Trichopherophyton* share many morphologic features with *Sawdonia ornata* [65], which was used as a source for missing details. The parameters chosen for the modelling of this plant are listed in table 4.14.

The modelling of *Trichopherophyton*’s growth is somewhat different from the other models due to its unique morphology. Its overall branching structure is similar, with axes that grow for a normally distributed period of time representing one plastochron, before bifurcating at an angle of $X \sim N(25, 6^2)^\circ$. Branching can be either dichotomous or pseudomonopodial. Unlike the other models, the axes grow in a loosely curled position, to shelter the developing fertile axes. Not enough of the plant’s structure is known to determine how the axes grew in space, so a positive orthotropic character is assumed, which is expressed after the branches unfurl. When the axes are first produced, they are 0.5 mm in width. As they mature, they thicken and usually reach a maximum diameter of up to 2.5 mm. The internode segments bear spinose projections ranging from $200\mu m$ to $1 \text{mm}$ in length.

The growth process continues while the apex’s vigour is greater than the branching threshold. However, once the vigour drops below that threshold, the apex produces a tighter fiddlehead-shaped structure that shelters the forming sporangia. Once the sporangia of the axis have matured, the entire plant unfurls. Since the positioning of the sporangia is unknown, a columnar arrangement similar to *Sawdonia ornata* was used in the model.

The rhizomatous axes of *Trichopherophyton* are very poorly understood, but they are
<table>
<thead>
<tr>
<th>Module</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germinator</td>
<td>delay</td>
<td>Places spore on the terrain, and tracks the delay before it germinates</td>
</tr>
<tr>
<td>Apex</td>
<td>age, length, vigour, fertile</td>
<td>Determines when and how the plant grows, branches, produces sporangia, and terminates</td>
</tr>
<tr>
<td>Vernation</td>
<td>age, length, vigour, fertile</td>
<td>Represents the tightly curled, fertile part of the plant, which unfurls after maturing</td>
</tr>
<tr>
<td>Internode</td>
<td>age, length, width</td>
<td>Represents internal sections of the aerial axes, and determines their length and width</td>
</tr>
<tr>
<td>Sporangium</td>
<td>(none)</td>
<td>Simple module that produces the sporangial stalk and places ‘HalfSporangium’ modules</td>
</tr>
<tr>
<td>HalfSporangium</td>
<td>age, length</td>
<td>Represents half of the body of a sporangium, to allow for dehiscence</td>
</tr>
<tr>
<td>Base</td>
<td>age, position</td>
<td>Stores the total age and position of the plant</td>
</tr>
</tbody>
</table>

Table 4.15: A list of the L-System modules used to model *Trichopherophyton*

assumed to be subterranean and were not modelled.

4.8.3 Implementation

As with the other models, the simulation begins with an axiom of one or more Germinator modules, which produce *Trichopherophyton* Base and Apex modules at a random locations on the terrain surface. The Apex grows in the same fashion as those of the other plants, but between each successive Internode module, a VernationBend module is placed which causes the axis to curl in on itself until it is ready to unfurl (Table 4.15). Axes branch either dichotomously or pseudomonopodially, with the same branching angle in both cases ($X \sim N(25, 6^2)$). Once the vigour drops below the branching threshold, the apex is replaced by a Vernation module.

The ‘Vernation’ module is responsible for creating the terminal circinate vernation structure of the plant, placing the sporangia in a column along it, and deciding when it unfurls. A string of ‘Internode’ modules is placed in the same fashion as the apex, but each of these modules is preceded by a ‘VernationBend’ module, which causes it to curl in a spiral. Meanwhile, sporangia are placed along the length of the fertile axis, in a phyllotactic spiral. After
Figure 4.49: The base functions used to define different aspects of *Trichopherophyton*’s form. 
A. *sporangiumWidthF*, an approximation of the longitudinal section of *Trichopherophyton*’s sporangia. 
B. *sporangiumSplitF*, causes the sporangia to rapidly but smoothly split open after some time. 
C. *vernationF*, dictates the angle at which each segment of the vernation curls, increasing over its length to create a tight spiral. 
D. *unfurlF*, controls the expression of the vernation angle over time.

Figure 4.50: Contour representing the transverse section of one valve of a *Trichopherophyton* sporangium
the sporangia have matured (reached the age at which their size is no longer increasing) the ‘VernationBend’ modules, whose angles are dictated by the ‘vernationAngle’ and ‘unfurl’ functions, cause the axis to straighten.

The ‘Sporangium’ modules are a placeholder, which immediately generate a sporangial stalk and two ‘HalfSporangium’ modules on separate branches. The ‘HalfSporangium’ modules utilize a semi-ellipsoid contour (Fig. 4.50) to give them their shape. The longitudinal contour of the sporangia is determined by the ‘sporangiumWidth’ function (Table 4.49). Depending on their age, a bend is applied to each ‘HalfSporangium’, away from the other half, causing them to dehisce.

During the interpretation step of the simulation, the modules described previously are translated into generalized cylinders. ‘Apex’ modules are rendered as a narrow cylinder tipped with a sphere, to terminate the axis if they are present. ‘Internode’ modules are rendered as cylinders of the prescribed width and length, and their colour is determined by the ‘colour’ function, based on their age. This step also renders the spinose projections of the internodes. Before rendering, the program checks whether or not the segments are within the view volume. If they are not, the rendering step is skipped to reduce processing time. The detail of the model is also determined by the camera’s distance from the ‘Base’ module. If the camera is far away, the polygon count is reduced by decreasing the number of sides per generalized cylinder. From a distance, ‘HalfSporangium’ modules are also modelled with fewer segments lengthwise due to their higher level of detail. The spinose projections of ‘Internode’ modules are small enough to be invisible at a distance, so the number of spinose projections rendered also depends on the camera’s distance from the axis.

The code for the *Trichopherophyton* L-System is given in Appendix A.6.
4.8.4 Results

Final renderings of *Trichopherophyton* are presented in this section. Fig. 4.51 illustrates *Trichopherophyton*’s development over time. Fig. 4.52 shows a mature plant. Fig. 4.53 shows details of *Trichopherophyton*’s axes and sporangia.

Figure 4.51: *Trichopherophyton* at different stages during its development.
Figure 4.52: A mature aerial axis of *Trichopherophyton* as it appears in the model.

Figure 4.53: A close-up of partially unfurled branches of *Trichopherophyton*. 
4.9 *Asteroxylon mackiei*

4.9.1 Description

*Asteroxylon* differs significantly from the other Rhynie chert plants in terms of both morphology and apparent growth habit. It is also the only Rhynie plant that may have extant relatives, in the form of lycophytes. It is relatively common, and one of the better studied plants at Rhynie. Its aerial axes are cylindrical and up to 12 mm thick [42]. Their surfaces are covered with leaf-like enations, each about 5 mm long, arranged in a phyllotactic spiral [43]. These were originally interpreted as leaves, but were later determined to lack the characteristics of true leaves. *Asteroxylon* was by far the tallest plant in its ecosystem, reaching heights of up to 40 cm, with a system of rhizomatous axes reaching depths of up to 20 cm. Second order branches emerged from the main axis modopodially. The angles of these branches are unknown due to poor preservation, so the branches’ connection to the plant is described as ‘lateral’. Higher order branches appear to have emerged both modopodially and dichotomously. The rhizomatous axes appear to have penetrated the substrate with a
branching structure similar to that of the root systems of extant plants.

The sporangia of *Asteroxylon* appear to have emerged laterally near the tip of the main axis. They were reniform and approximately 7.0 mm by 2.5 mm in cross-section [48]. Further details on the structure of the sporangia are scarce due to poor preservation and lack of specimens in anatomical connection with the axes.

*In situ* specimens of *Asteroxylon* are always found in association with other Rhynie plants, most commonly *Nothia*, *Rhynia*, and *Aglaophyton*. This suggests that *Asteroxylon* was part of the plant community, rather than an early colonizer [58]. It is likely that *Asteroxylon* was more tolerant of dry conditions than other Rhynie plants, due to its extensive subterranean ‘rhizome’ system, and more efficient transpiration mechanisms [79].

Several previous reconstructions of *Asteroxylon* are available, and are detailed in Fig. 4.56.

### 4.9.2 Modelling

The aerial axes grow for a normally distributed period of time representing one plastochron, before producing a modopodial branch of higher order. New axes emerge at an angle in the range of \( X \sim \mathcal{N}(50, 5^2) \)° to the parent axis. Based on their structure, it is assumed that *Asteroxylon* exhibits a positive orthotropism in its main axes, with weaker effect in higher
order branches.

The internodes of *Asteroxylon* are its greatest point of interest due to the leaf-like enations that are unique within this ecosystem. The enations were modelled as a number of leaf-shaped surfaces, which are arranged in a phyllotactic spiral around the internodes. Enations emerge from their parent axis at an angle within $X \sim N(5, 1^2)$ °.

The rate at which each axis grows, if and when it branches, and when it terminates is controlled by the vigour of that axis’ apex. The plant may branch as long as the vigour is above a fixed threshold. Once the vigour drops below that threshold, the main axis has a chance to produce a sporangium during each time-step. Over time, a scattering of sporangia will be produced near the tip of the main axis. When axes are first produced, they are 0.5 mm in width. As they mature, they thicken and usually reach a maximum diameter of up to 12.0 mm for the main axes, decreasing in the higher order axes.

The rhizomatous axes were not modelled because they are subterranean (Future work: simplified rhizomes could aid in positioning the aerial axes, as they do with *Nothia*).
<table>
<thead>
<tr>
<th>Module</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germinator</td>
<td>delay</td>
<td>Places spore on the terrain, and tracks the delay before it germinates</td>
</tr>
<tr>
<td>Apex</td>
<td>age, length, vigour, order</td>
<td>Determines when and how the plant grows, branches, produces sporangia, and terminates</td>
</tr>
<tr>
<td>Internode</td>
<td>age, length, width, order</td>
<td>Represents internal sections of the aerial axes, and determines their length and width</td>
</tr>
<tr>
<td>Enation</td>
<td>age, scale, vigour, order, angle</td>
<td>Represents one enation of the plant, including its size and shape.</td>
</tr>
<tr>
<td>Sporangium</td>
<td>angle</td>
<td>Represents a single sporangium of the plant</td>
</tr>
<tr>
<td>Base</td>
<td>age, position</td>
<td>Stores the total age and position of the plant</td>
</tr>
</tbody>
</table>

Table 4.16: A list of the L-System modules used to model *Asteroxylon*

<table>
<thead>
<tr>
<th>Function</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>vigourDecay(order)</td>
<td>$1 - factor \cdot [1 - vigourDecayF(X \sim U([0,1]))^a] \cdot (order + 1) \cdot dt$</td>
</tr>
<tr>
<td>width(vigour,order)</td>
<td>$\frac{widthF(1 - vigour) \cdot orderReductionF\left(\frac{order}{maxOrder}\right)}{factor}$</td>
</tr>
<tr>
<td>scale(vigour,order)</td>
<td>$\frac{scaleF(1 - vigour) \cdot orderReductionF\left(\frac{order-1}{maxOrder}\right)}{factor}$</td>
</tr>
<tr>
<td>enationBend(vigour)</td>
<td>$enationBendF(1 - vigour)$</td>
</tr>
<tr>
<td>elasticity(order)</td>
<td>$(maxElasticity - order \cdot factor) \cdot dt$</td>
</tr>
<tr>
<td>ageFactor(age)</td>
<td>$ageWidthF\left(\frac{age}{factor}\right)$</td>
</tr>
<tr>
<td>ageBend(age)</td>
<td>$ageBendF\left(\frac{age}{factor}\right)$</td>
</tr>
<tr>
<td>bend(vigour,age)</td>
<td>$enationBend(vigour) + ageBend(age)$</td>
</tr>
<tr>
<td>enationGrowth(age)</td>
<td>$enationGrowthF\left(\frac{age}{factor}\right)$</td>
</tr>
</tbody>
</table>

Table 4.17: Functions used to model different aspects of *Asteroxylon*. $a$, $factor$, and $maxOrder$ are constants. See Fig. 4.57 for base functions.

4.9.3 Implementation

*Asteroxylon* begins its life in the same way as the other models, with a Germinator module that produces a Base and Apex module. The apices of *Asteroxylon* store the order of branching as an additional parameter (Table 4.17). The order of branching controls the length of each branch, the rate of vigour decay, and the strength of the plant’s orthotropism. Higher order branches are shorter, have faster decaying vigour, and a weaker tropism.

Once an Apex with order zero has produced a fixed number of Internode modules, it produces a modopodial branch. The new Apex has all parameters zero except for vigour,
which is equal to that of its parent Apex. Branches of higher order apices can be monopodial or dichotomous. While the vigour is below the branching threshold, sporangia will start to be produced in a column around the main axis. Sporangium generation is more likely near the apex. Sporangium modules will appear around the main axis in a phyllotactic spiral, branching off of the axis at an angle of $X \sim N(30, 3^2) \, ^\circ$. Axes terminate when the vigour drops below the termination threshold.

‘Internode’ modules of Asteroxylon are handled differently than those of other plants due to its more complex structure. The width of the axes is explicitly defined based on the age, vigour, and position of the axis, which are used as inputs to the ‘ageFactor’ and ‘width’ functions (Table 4.17). Each segment is also responsible for producing its associated enations. Length and age are handled in a fashion similar to the other plants. They age by a factor of $dt$ each time step, and grow in length according to the ‘growthRate’ function.

During the interpretation step of the simulation, the modules described previously are translated into viewable generalized cylinders or surfaces. ‘Internode’ modules are rendered as cylinders of the prescribed width and length, and their colour is determined by the ‘colour’ function, although they are mostly hidden behind their covering of enations. ‘Enation’ and ‘Sporangium’ modules make up most of the model’s outer appearance. Both of these are represented by Bézier surfaces. Before rendering, the program checks whether or not the segments or surfaces are within the view volume. If they are not, the rendering step is
Figure 4.57: The base functions used to define different aspects of *Asteroxylon*’s form. 

A. *ageBendF*, defines the angle of each enation over time, causing them to ‘unfold’ from the apex. 

B. *ageFactor*, affects the width of the internodes over time. 

C. *enationBendF*, together with *ageBendF*, defines the angle of the enations. 

D. *enationGrowthF*, defines the size of the enations over time. 

E. *orderReductionF*, used in other functions to reduce the size of enations and axes of higher order branches. 

F. *widthF*, an approximation of the longitudinal section of an axis, defining the width over the length of modelled axes.
skipped to reduce processing time, which is especially significant in this model due to the number of ‘Enation’ surfaces. The detail of the model is also determined by the camera’s distance from the ‘Base’ module. If the camera is far away, the polygon count is reduced by decreasing the number of sides per generalized cylinder.

The code for the *Asteroxylon* L-System is given in Appendix A.7.

4.9.4 Results

Final renderings of *Asteroxylon* are presented in this section. Fig. 4.58 shows a mature plant. Fig. 4.59 shows details of *Asteroxylon*’s axes and sporangia. Fig. 4.60 illustrates *Asteroxylon*’s development over time.

![Image of a model of a single aerial axis of *Asteroxylon*]

Figure 4.58: A model of a single aerial axis of *Asteroxylon*. 
Figure 4.59: A close-up of the main axis of *Asteroxylon*.

Figure 4.60: *Asteroxylon* at different stages during its development.
Chapter 5

Ecosystem Modelling

5.1 Environmental Considerations

The secondary goal of this thesis work was to reconstruct the ecosystem at the Rhynie chert using the seven models from Chapter 4 and palaeoecological data regarding these plants. Palaeobotanists have hypothesized that various aspects of the environment affect the success of different Rhynie chert plant species [58] [80]. To model the ecosystem of the Rhynie chert, I chose three conditions to affect each plant’s chances: the availability of water, prevalence of sand and sinter in the substrate, and abundance of soil and plant litter in the substrate. These will be shortened to water, sinter, and litter for ease of reference. Factors such as the chemistry and temperature of the water and soil have been shown to affect plant growth in extant hot spring systems, but information regarding these factors in the Rhynie chert is too limited to be meaningful [80].

The conditions and location of deposition of each plant fossil were used to choose reasonable parameters for each plant’s ideal environment (Table 5.1). The water, sinter, and litter parameters are specified in the range of [0, 1] at each point on the terrain surface, and each plant has a smaller range specified for each condition, which represents its ideal growth environment. Given the input of parameter choices and terrain information, a plant community is generated over time, reflecting the characteristics of the plants as observed in the fossil record. This includes the plants associations with other plants, their tendency to form monotypic stands, and the frequency with which they appear.
Table 5.1: The ideal ranges for each simulation parameter for each plant. Minimum and maximum values can be interpreted as the minimum required conditions and the maximum tolerated conditions. If all parameters are inside the ideal range, the plant is healthy. The plant suffers if any parameter is outside the ideal range, and will fail to germinate on substrates that are too hostile.

5.2 Parameter Selection

*Horneophyton*

Palaeobotanists have suggested that *Horneophyton*’s preferred living conditions were damp to wet environments [81], so its water preference was chosen to be 0.6 to 1.0. *Horneophyton* fossils in life position are often found in monotypic stands, suggesting that *Horneophyton* was a colonizer of sinter substrates [81]. In other chert beds, it is found in association with other Rhynie plants (but rarely with *Rhynia*), suggesting that it successfully competed with other species later in the plant succession [58]. Combining these two factors, a sinter tolerance of 0.1 to 0.7 was selected. Its litter requirement was set at 0.4 to 0.8 due to the tendency of *Horneophyton*’s rhizomatous axes to cut through both plant litter and sandy soils [81]. This means that in wet, high sinter environments the plant will likely appear in monotypic stands, whereas in more hospitable environments it will compete with the other genera.

*Asteroxylon*

*In situ* specimens of *Asteroxylon* are always found in association with other Rhynie plants, most commonly *Nothia, Rhynia* and *Aglaophyton*. This suggests that *Asteroxylon* was part of the plant community, rather than an early colonizer of sinter substrates [58]. Its sinter
tolerance was set at 0.0 to 0.3 to reflect this. Its litter preference at 0.5 to 1.0 to reflect the tendency for its penetrating rhizomatous axes to prefer litter. It is likely that Asteroxylon was more tolerant of dry conditions than other Rhynie plants, due to its extensive subterranean ‘rhizome’ system, and more efficient transpiration mechanisms [79]. Its water preference was set to 0.1 to 0.6 to reflect this.

*Nothia*

Information on the palaeoecology of *Nothia* is somewhat limited because the aerial axes are rarely found in life position. However, the rhizomatous axes are occasionally found in monotypic assemblages, or penetrating litter of other plants, commonly *Asteroxylon*. This suggests that *Nothia* grew in sandy litter substrates, and preferred similar environments to *Asteroxylon* [82]. Considering this, the water preference for *Nothia* was set to 0.2 to 0.6. Sinter tolerance is 0.0 to 0.4, and litter preference is 0.3 to 0.9.

*Ventarura*

Because *Ventarura* fossils are often found transported, and not in life position, it is difficult to make conclusions about their original habitat. However, it seems likely that *Ventarura* lived on or near the edges of the bodies of water that transported its remains [85]. Water preference for *Ventarura* is thus set at 0.9 to 1.0. Due to the presence of apparently *in situ* rhizomatous axes, it is assumed that *Ventarura* preferred both organic-rich and sandy substrates [85]. Sinter tolerance was set to 0.0 to 0.5 and litter preference to 0.5 to 1.0 to reflect this.

*Trichopherophyton*

Although specimens of *Trichopherophyton* are rare and poorly preserved, their locations are telling as to its possible palaeoecology. Fossils are found both in growth position and in plant litter, and they are always associated with other plants, such as *Nothia*, *Horneophyton* and *Rhynia*, suggesting that *Trichopherophyton* was a late colonizer of humic substrates [58]. A
low sinter tolerance of 0.0 to 0.2 and a litter preference of 0.7 to 1.0 were chosen to reflect this. Other than its associations with other plants, nothing is known about Trichopherophyton’s preference for moisture, thus a water preference of 0.4 to 0.8 was chosen to reflect these associations.

**Aglaophyton**

*In situ* fossils of Aglaophyton indicate that it preferred to grow on litter-covered, organic-rich substrates rather than sinter. Its sinter tolerance was set to 0.0 to 0.2, and its litter preference to 0.6 to 1.0. The plant also seems to be drought resistant, due to adaptations that allow it to conserve water [58]. However, Aglaophyton is also associated with *in situ* algae, crustaceans, and fungi, however, suggesting that it was exposed to wet conditions during its life cycle, possibly during floods. Evidence suggests that Aglaophyton spores required wet conditions to germinate [67]. In the simulation, Aglaophyton was given a wide water preference, ranging from 0.2 to 0.8. Its tolerance to various conditions causes it to be found both in monotypic stands and in association with all other common Rhynie plants.

**Rhynia**

Specimens of Rhynia found in growth position are often found isolated from other Rhynie plants, suggesting that it grew in monotypic stands, and was therefore an early colonizer of sinter substrates [83]. Its tolerance of these substrates is 0.0 to 0.7, and its preference for litter it 0.1 to 0.7. Rhynia is found in association with all other Rhynie plants in other deposits however, suggesting that it was highly tolerant and could withstand competition [58]. Its water preference is set to 0.2 to 0.6 to reflect its apparent preference for well-drained substrates [83].
Figure 5.1:  
A. The texture-mapped terrain viewed from above, with colours representing the actual appearance of the substrate. The area shown is approximately five metres square. This terrain was generated using EarthSculptor [75].  
B. An image representing the availability of resources across a section of the above terrain. Each colour represents the availability of one resource from 0 to 1. Water is represented by blue, litter by green, and sinter by red. The black regions represent areas where spores cannot germinate (a moving stream and a hot geothermal pool). Fig. 5.2 shows a visual representation of the each plant’s preference for these colours.
5.3 The Ecosystem Simulation

In each simulation, an image (created by hand in Adobe Photoshop) is used to indicate the availability of each resource across the terrain (Fig. 5.1). A number of spores of each genus are placed randomly on the terrain and the plants begin to grow. The vigour value of each plant is determined by its compatibility with the terrain it germinated on. Lower vigour causes the plant to grow slower and die sooner. If the plant survives to maturity, it spreads spores to its surroundings based on its vigour value, to represent greater fertility in healthier plants. These spores continue the cycle by producing new plants if the substrate is suitable, but dying off immediately if their initial vigour is too low. This creates a community of plants that thrive in regions where the conditions are right for their growth (Fig. 5.3).

Growing plants can also interact with each other in the system. Each plant is represented by a circle, which denotes the area influenced by the plant. The size of each plant’s area of

\[\text{Aglaophyton}\]
\[\text{Asteroxylon}\]
\[\text{Horneophyton}\]
\[\text{Nothia}\]
\[\text{Rhynia}\]
\[\text{Trichopherophyton}\]
\[\text{Venturura}\]
influence is an approximation of the size of the plant itself, and its effect on the available space, light and resources in its vicinity. The size of this region is determined by the plant’s species, age, and vigour. When two plants collide, the one with a smaller area of influence becomes dominated and dies. This simulates plant thinning and allows more suitable plants to become dominant under the right conditions [45].

The code for the environmental simulation L-System is given in Appendix A.8.

5.4 Results

After being modelled separately, the plant models and environmental model were combined to create an image of the plants in context. The positions of the plants were exported from the environmental simulation, and full-detail plant models were grown on the terrain in their place. The result is a visualization of what the ecosystem at the Rhynie chert may have looked like. Figures 5.4, 5.5 and 5.6 are screen captures of the final models. These models were generated and rendered using lpfg.
Figure 5.3: An example of a simplified ecosystem after 5, 100, 500 and 1500 iterations. Each of the seven species are present, represented by coloured circles. Species often fail to compete under certain local environmental parameters, and form sustainable populations in more suitable regions. The terrain, and its colour representation, are identical to that shown in Fig. 5.1.
Figure 5.4: A close-up of a dense stand of Rhynie chert plants, predominantly (1) *Aglaocephylon* and (2) *Asteroxylon*.
Figure 5.5: An overhead view showing a sparse but diverse community of plants. (1) *Aglaoiphyton*, (2) *Asteroxylon*, (3) *Horneophyton*, (4) *Nothia*, (5) *Rhynia*, (6) *Trichopherophyton*, (7) *Ventarura*
Figure 5.6: A different angle on a denser plant community, comprising predominantly (1) *Aglaophyton*, (2) *Asteroxylon*, (3) *Nothia*, (4) *Rhynia*, and (5) *Trichopherophyton*. 
Chapter 6

Discussion

6.1 Plant Models

The goal of this thesis work was to realistically reconstruct extinct plants using existing plant modelling techniques and evaluate their suitability for this purpose. This was accomplished by modelling each of the plant species discovered at the Rhynie chert. Many basic aspects of these plants could be reproduced easily. Repetitive elements such as bifurcating branching structures are easily modelled by L-systems. However, L-systems describe plants at the level of modules, discretizing them into components such as ‘Sporangium’ or ‘Aerial Apex’. This modular approach breaks down when attempting to model minuscule details such as Aglaophyton’s spores or Trichopherophyton’s spines. Likewise, the connection between axes and sporangia often appears too abrupt.

The plant models presented in this thesis are intended to be the most accurate to date. Serious flaws in previous models – such as inaccurate sporangium shapes and dimensions, and incorrect axis widths – have been identified and corrected. Additionally, these models are the first dynamic models of Rhynie chert species, and can be used to create limitless specimens of the same species under different conditions, and animate their entire growth process.

Each of the models, with the exception of Trichopherophyton, was constructed while considering previous reconstructions of the plant in question. Of primary importance in previous reconstructions are hypotheses on how the fragmentary components of each plant fossil were actually connected in life. Useful information from previous reconstructions was then combined with newly published results and observations of fossil evidence. This is often a difficult task, because source data (in the form of direct fossil evidence) are not always
adequately described. Cross-referencing direct fossil evidence helped to avoid some of the pitfalls of previous reconstructions, such as compounding mistakes by redrawning solely from other reconstructions.

The model of *Horneophyton* was based strongly on previous reconstructions of this plant. The main contributions, beyond the dynamic nature of the model, were the improvements on scale and shape when modelling the sporangia and rhizomatous tubers. Accurate scaling of axis width and total height was also a priority.

The *Aglaophyton* model also expands on detailed previous models of the same plant, by improving details such as the scale of sporangia and axes. The dehiscence and subsequent desiccation of the sporangia is also animated over time. Another contribution is the dynamic interaction between the plant’s decumbent axes and the ground. Previous models simply represented the decumbent axes at maturity, while in this model, the final position emerges as a result of collisions with the terrain during growth, assuming that these axes grew in decumbent position.

Previous models of *Rhynia* focussed on the aerial axes and sporangium placement. The model in this thesis incorporates those elements and introduces dynamic growth, also including interaction between the decumbent axes and the ground.

The focus on the model of *Nothia* was to accurately capture the irregular surfaces of the aerial axes, and the complex arrangements of the sporangia. The subterranean rhizomatous axes of *Nothia*, although not visible above the terrain, were modelled to distribute the aerial axes in a logical fashion.

The model of *Ventarura* is, to my knowledge, the first model of an entire plant of this species. All known details about the plant were incorporated into this model. Nevertheless, more assumptions are made and the final model is unlikely to be entirely accurate.

To my knowledge, the model of *Trichopherophyton* is the first model of this plant ever attempted. Because of limited fossil evidence, the accuracy of the model is likely also limited.
Many details of this plant were based on a better known related plant called *Sawdonia ornata*. However, the known information about *Trichopherophyton*, such as the nature of its spines and sporangia was included. A hypothetical model regarding how this plant’s vernations may have unfurled was also incorporated.

The final model of *Asteroxylon* is similar to previous reconstructions, and the main contributions of this model are the dynamic growth and sporangium placement.

Reconstruction of fine details of these extinct plants makes the final models very large. As a result, modelling the plants with view-dependent level of detail becomes very useful. Different elements of the plants can be rendered in reduced detail from a distance, or in the case of small details, not at all. Modules that are outside the view volume can also be ignored.

### 6.2 Ecosystem Model

The environmental model combines the sparse data about the palaeoecology of the Rhynie chert ecosystem, and attempts to combine it into a reconstruction of the interaction between Rhynie chert plants. This is, to my knowledge, the first dynamic reconstruction of an ancient ecosystem.

The method of using areas of influence (or other influence kernels) was already established. A parametrization of environmental conditions was established in [13], but different parameters were used to suit the available data and the character of the ancient Rhynie chert ecosystem. This type of parametrization is well suited to sparse palaeoecological data due to their imprecise nature.

In fossil deposits, *Aglaophyton* was found to associate mainly with *Nothia, Asteroxylon*, and *Horneophyton*, and occasionally *Rhynia*. By choosing logical environmental preferences based on palaeoecological data, these associations are accurately reflected in the model.

Fossil evidence suggests that *Asteroxylon* was mainly found in association with other
plants rather than in monotypic stands. It mainly associated with *Nothia*, *Rhynia* and *Aglaoophyton* and occasionally *Ventarura*. These interactions are fairly well represented in the model. Improvements could be made if *Asteroxylon*’s interactions with other plants were better understood. Being the largest plant in the ecosystem, it likely interacted differently (perhaps slower growing but harder to dominate), which is not taken into account by the current model.

*Horneophyton* often grew in monotypic stands, but also grew in association with many other *Rhynie* chert plants. In the fossil record, it is rarely found in association with *Rhynia*. In the model, *Horneophyton* grows both in monotypic stands and in association with other plants depending on the conditions, as expected. However, it is found in the same stand as *Rhynia* much more often than it should. This could hint at environmental parameters (e.g. soil acidity and texture) not taken into account by the model due to lack of evidence.

The fossil record suggests that *Nothia* grew both in monotypic stands and in association with other plants, especially *Asteroxylon*. This seems to hold true in the model, with *Nothia* growing in monotypic stands under certain conditions, and together with a diverse population in more generally favourable conditions.

*Rhynia* was found in association with all other Rhynie plants, but rarely with *Horneophyton*. In the ecosystem model, *Rhynia*’s interactions seem fairly consistent with fossil data, with the exception of its interaction with *Horneophyton*, as mentioned above.

Little is known about the palaeoecology of *Trichopherophyton* or *Ventarura*. They seem to have associated somewhat with all other Rhynie chert plants, which is also true in the model.

In general, the associations and relative quantities of each plant were close to expectations. Improvements could be made given more evidence. Computer renderings of the ecosystem containing hundreds of individual plants (based on the models presented in chapter 5) are the final result of this effort (Figs. 5.4, 5.5 and 5.6).
6.3 Modelling Challenges

In general, modelling the Rhynie chert plants with L-systems worked quite effectively. Generalized cylinders are well suited to modelling the repetitive branching axes of these ancient plants, as well as many of their organs. However, some plant organs cannot adequately be modelled as generalized cylinders. In the case of *Horneophyton* sporangia, modelling their branching structure as a implicit surface based on a skeleton would greatly improve their accuracy. Functionality for implicit surfaces would have to be added to lpfg for this purpose.

Realistic connectivity between plant organs is also difficult to capture with L-systems, due to the modular representation of plant components in L-systems. The plants would likely have to be remodelled in a different manner to overcome this shortfall.

*Trichopherophyton*’s spines were difficult to model, because they are so numerous. The use of volumetric textures or geometry shaders would be more efficient than using hundreds of generalized cylinders.

6.4 Future Work

In the model of *Aglaophyton*, axes are grown in decumbent position, however, another hypothesis states that they grew in an upright position and collapsed under their own weight repeatedly. Modelling this alternate growth strategy should be the subject of future work.

The ecosystem model could also be greatly improved by allowing plants to modify the environment as they grow. For example, early colonizers are thought to grow on litter-poor substrates, while others require richer soil to germinate. Litter accumulates over time due to plant death, and therefore, plants in the simulation should increase the litter within their sphere of influence upon death. To compensate for the increased litter quantities, geyser eruptions would need to be modelled to add sinter to the environment and remove litter (and possibly kill plants as well).

The spheres of influence themselves are an over-simplification of the plants’ interactions
with one another, especially when dealing with spreading species like *Aglaophyton* and *Rhynia*. The spheres of influence for these plants could be made more accurate by giving a separate sphere to each aerial axis, instead of the plant as a whole. This could also be used to monitor the success of different shoots individually, and allow some parts of a spreading plant to thrive while others become dominated.

Many ecosystem models take into account the facilitative effect of other plants when placing new plants [12]. Due to lack of evidence, facilitation was not included in this ecosystem model. However, the propensity for some plants to grow in the same communities with other species could constitute a positive interaction between the two species. With more information on the interactions of these ancient plants, facilitation could be introduced into the ecosystem model.
Bibliography


Robert Kidston and William Henry Lang. Xxxiii. on old red sandstone plants showing structure, from the rhynie chert bed, aberdeenshire. part v. the thallophyta occurring in the peat-bed; the succession of the plants throughout a vertical section of the bed, and the conditions of accumulation and preservation of the deposit. *Transactions of the Royal Society of Edinburgh*, 52(33):855–902, 1921.


Appendix A

L-System Code

A.1 *Horneophyton*

```c
#include <algorithm>
#include <cstdio>
#include <string>
#include <vector>
#include <iostream>
#include "lpfgall.h"
using namespace std;

#define nran(mean, sd)
    ((sqrt(-2*log(ran(1)))*cos(6.283185307*ran(1)))*sd+mean) // Normally distributed random number with given mean and sd
#define ranb(low, high) (ran((high)-(low))+(low))

#define WTSZ (80) // Water size
#define WTHT (1.5) // Water height
#define RNGT (75) // Range on terrain where germinator can appear; max 75 to stay on terrain at scale 5
#define DEATH 40 // Base age at which the plant dies
#define DFAC (25*cameraPos.scale) // Distance factor for lod
#define DELT 0.08 // Non-Varying Timestep

#define HRN_NNAG (0.24) // Age to generate a new node
#define HRN_SPAG (0.16) // Age to generate sporangium together with other conditions
```
# define HRN_Brag (12) // Age to branch
# define HRN_Bang nran(22,8) // Branching Angle
# define HRN_Dang nran(0,5) // Deflection Angle
# define HRN_Rrot nran(137.5/HRN_Brag,5) // Roll Branches
# define HRN_Sang nran(30,6) // Sporangium angle
# define HRN_IvIG nran(1.2,0.3) // Initial vigour value
# define HRN_SbNG nran(45,10) // Sporangium Branching Angle
# define HRN_IsVG nran(0.195,0.008) // Initial sporangium 'vigour’ value
# define HRN_SrrT nran(27.5,3) // Roll Sporangium Lobes
# define HRN_RhsG (13.0) // Number of segments in rhizome
# define dt nran(DELT,DELT/2.0) // Varying Timestep

#define growthRate(age) (func(growthRateF,age)*dt*nran(0.65,0.15))
#define colour(a) floor(64+71*func(colourF,(a/47.512)))
#define colourS(a) floor(140+71*func(colourF,(a/47.512)))
#define colourR(a) floor(64-27*func(colourF,(a/47.512))) // ????????
#define vigourDecay(yh)
   (1.0-((1.0-pow(func(vigourDecayF,ran(1))*(pow(yh,0.1f)),0.05f))*dt*25.0))
#define rhizomeWidth(n) (func(rhizomeWidthF,(n+1)/HRN_RhsG)*3.0)
#define shrivel(age) func(shrivelF,(age-DEATH)/20.0)
#define lod(dist) (6-floor(6*func(lodF,(dist/DFAC))))

VisibilityMode terrainVisibilityMode;
CameraPosition cameraPos;
float fov;
int steps;

module Yardstick(int);
module YardstickI(int);
module Water();
module HRNGerminator(int); //
Horneophyton Germinator (Delay)
module HRNApex(float, float, int, float); //

Horneophyton Apex (Age, Length, NumberOfNodes, Vigour)
module HRNInternode(float, float, float, float, float); //

Horneophyton Internode (BaseAge, Age, Length, Vigour, Width)
module HRNSporangium(float, float, int, float); //

Horneophyton Sporangium (Age, Length, NumberOfNodes, Vigour)
module HRNSporangiumNode(float, float, float, float, float); //

Horneophyton Sporangium Node (Age, Length, Vigour, Width)
module HRNRhizome(float, float, int, float); //

Horneophyton Rhizome (Age, Length, NumberOfNodes, Vigour)
module HRNRhizomeInterior(float, float, float, float, float); //

Horneophyton Rhizome Interior (Age, Length, Vigour, BaseWidth, WidthFactor)
module HRNBase(float, V3f); //

Horneophyton Base (Age, Position)

bool InsideView(V3f point) {
//return true; // Speed comparison with InsideView off

V3f camDirReverse = point - cameraPos.position;
camDirReverse.Normalize();
return cameraPos.head*camDirReverse > abs(cos(fov/1.8));
}

float DistanceToCamera(V3f point) {
return sqrt(pow(point.x - cameraPos.position.x, 2.0f) + pow(point.y - cameraPos.position.y, 2.0f) +
}

Start: {
cameraPos = GetCameraPosition(0);
terrainVisibilityMode = Shaded;
terrainVisibilityAll(terrainVisibilityMode);
steps = 0;
}

StartEach: {
cameraPos = GetCameraPosition(0);
fov = 2.0 * atan(1.0/cameraPos.scale * tan(M_PI/4.0));
steps++;
}

derivation length: 300;
Axiom: SetColor(12) Terrain(cameraPos) Water() /*SB() Yardstick(0) EB()*/
CurrentContour(1) CurrentTexture(2) SB() HRNGerminator(3) EB();

/*********************/
production:

Water(): {
produce SetColor(5)
SP() MoveTo(-WTSZ,WTHT,-WTSZ) PP() MoveTo(WTSZ,WTHT,-WTSZ) PP()
MoveTo(WTSZ,WTHT,WTSZ) PP() MoveTo(-WTSZ,WTHT,WTSZ) PP() EP();
}

// A ruler for measuring plant height (and width)
Yardstick(c): {
if (c == 0) {
V3f pointOnTerrain;
V3f initPoint = V3f(-5,0,-5);
bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
if (hitTerrain) {
Printf("Yardstick Point: %f %f
 %f\n", pointOnTerrain.x, pointOnTerrain.y, pointOnTerrain.z);
produce MoveTo3f(pointOnTerrain) Yardstick(1);
} else {
Printf("Point of FAILURE: %f %f
 %f\n", initPoint.x, initPoint.y, initPoint.z);
Printf("Someone dropped the yardstick off a cliff\n");
produce ;
}
} else if (c <= 20) {
produce YardstickI(c) Yardstick(c+1);
}

// Drops a spore randomly on the terrain
HRNGerminator(delay): {
if (delay > 0) produce HRNGerminator(delay -1);

V3f pointOnTerrain;
V3f initPoint = V3f(ranb(-RNGT,RNGT),0,ranb(-RNGT,RNGT));

initPoint = V3f(0,0,0); // TEST

bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
pointOnTerrain.y -= nran(1.1,0.1);
if (pointOnTerrain.y < WTHT) {
Printf("Horneophyton spore drowned at: %f %f
 %f\n", initPoint.x, initPoint.y, initPoint.z);
return;
}
if (hitTerrain) { // Success!!
float vigour = HRN_IVIG;

133
Printf("Horneophyton germinated at: %f %f %f Vigour: %f, \\
        (%i)\n",pointOnTerrain.x,pointOnTerrain.y,pointOnTerrain.z,vigour,steps);
produce RollR(ran(360)) HRNBase(0, pointOnTerrain) StartGC()
    HRNRhizome(0,0,0,vigour) GetHead(0,0,0);
} else { // Failure!!
    Printf("Point of FALIURE: %f %f \\
            %f
",initPoint.x,initPoint.y,initPoint.z);
    Printf("The spore was falling to the ground, but then it saw something so \\
            surprising that it missed\n");
    produce SetColor(23) MoveTo3f(initPoint) Sphere(1) F(3) Sphere(1) F(3) \\
            Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) \\
            Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1); // \\
            Test
}
}

consider: GetHead HRNBase;

HRNBase(baseAge, pos) < HRNApex(age, length, n, vigour) > 
    GetHead(xh,yh,zh): {
        if (vigour < 0.25) return; // Terminate
        if (vigour < 0.35 && vigour > 0.3 && age >= HRN_SPAG && ran(1) < 0.15) {
            // Produce Sporangium TODO link check to dt
            produce HRNSporangium(0,0,2,HRN_ISVG) GetHead(0,0,0) GetPos(0,0,0);
        } else if (n >= HRN_BRAG && vigour > 0.35) { // Bifurcate
            produce HRNInternode(baseAge, age,length,vigour,0.05) GetPos(0,0,0) SB()
                Left(HRN_BANG) HRNApex(0,0,0,vigour) GetHead(0,0,0) EB()
                SB() Right(HRN_BANG) HRNApex(0,0,0,vigour) GetHead(0,0,0) EB();
        } else if (age >= HRN_NNAG) { // Start a new segment
            produce HRNInternode(baseAge, age,length,vigour,0.05) GetPos(0,0,0)
                RollR(HRN_RROT) Up(HRN_DANG) HRNApex(0,0,n+1,vigour);
        }
}
produce HRNApex(age + dt, length + growthRate(age) * min(vigour, 1.0f), n, vigour * vigourDecay(yh));
}

// Pipe model
consider: HRNInternode;

HRNInternode(baseAge, age, length, vigour, width) > SB() HRNInternode
    (i0, i1, i2, i3, widthRL) EB() SB() HRNInternode (i4, i5, i6, i7, widthRR) EB(): {
    float deltaT = dt;
    produce HRNInternode(baseAge + deltaT, age + deltaT, length + growthRate(age) * min(vigour, 1.0f), vigour, sqrt(pow(widthRL, 2.0f) + pow(widthRR, 2.0f)));
}

HRNInternode(baseAge, age, length, vigour, width) > HRNInternode (i0, i1, i2, i3, widthR): {
    float deltaT = dt;
    produce HRNInternode(baseAge + deltaT, age + deltaT, length + growthRate(age) * min(vigour, 1.0f), vigour, widthR * 1.01);
}

HRNInternode(baseAge, age, length, vigour, width): {
    float deltaT = dt;
    produce HRNInternode(baseAge + deltaT, age + deltaT, length + growthRate(age) * min(vigour, 1.0f), vigour, width);
}

consider: GetHead;

HRNSporangium(age, length, n, vigour) > GetHead(xh, yh, zh): {
    if (vigour < 0.15) return;
    int numSec = 0.03 / pow(vigour, 2.5f);
if (n >= numSec && vigour > 0.18 && ran(1) < 0.9) {
    produce HRNSporangiumNode(age, length, vigour, 0.05) GetPos(0, 0, 0) SB()
        Left(HRN_SBNG) HRNSporangium(0, 0, 0, vigour - 0.01) GetHead(0, 0, 0) EB()
        SB() Right(HRN_SBNG) HRNSporangium(0, 0, 0, vigour - 0.01) GetHead(0, 0, 0) EB();
} else if (n <= 4 && age >= 0.03) {
    produce CurrentContour(min(n+2, 4))
        HRNSporangiumNode(age, length, vigour, 0.05) GetPos(0, 0, 0)
        RollR(HRN_SRRT) Up(HRN_DANG * 1.5) HRNSporangium(0, 0, n+1, vigour);
}

produce HRNSporangium(age+dt, length+growthRate(age)*vigour*0.5, n, vigour*vigourDecay(yh));

HRNSporangiumNode(age, length, vigour, width): {
    float deltaT = dt;
    produce HRNSporangiumNode(age+deltaT, length+growthRate(age)*vigour*0.5, vigour, width+growthRate(age)*vigour);
}

HRNRhizome(age, length, n, vigour): {
    if (n > HRN_RHSG) produce GetPos(0, 0, 0) HRNApex(0, 0, 0, vigour);
    else if (length > 0.005) {
        float contour = (n*2.99/HRN_RHSG)+5;
        int contourID = contour;
        float contourBlend = contour - contourID;
        produce BlendedContour(contour, contour+1, contourBlend)
            HRNRhizomeInterior(0, length, vigour, rhizomeWidth(n), 0)
            HRNRhizome(0, 0, n+1, vigour);
    }
    produce HRNRhizome(age+dt, length+growthRate(age)*vigour*0.2, n, vigour);
consider: HRNInternode;

HRNRhizomeInterior(age, length, vigour, baseWidth, widthFactor) > 
    HRNInternode (i0, i1, i2, i3, widthR): {
    float deltaT = dt;
    produce HRNRhizomeInterior(age+deltaT, length+growthRate(age)*vigour*0.1, 
        vigour, baseWidth, widthR);
}

HRNBase(age, pos): {
    produce HRNBase(age+dt, pos);
}

/**********************************
interpretation:
maximum depth: 2;

consider: GetPos;

YardstickI(c): {
    if (c%10 == 0) produce SetWidth(1) SetColor(15) F(1);
    else produce SetWidth(1) SetColor(22+c%2) F(1);
}

HRNApex(age, length, n, vigour): {
    produce SetWidth(0.04) F(length) EndGC() Sphere(0.02);
}

HRNInternode(baseAge, age, length, vigour, width) > GetPos(x,y,z): {
    if (InsideView(V3f(x,y,z)))
    produce SetColor(colour(age)) SetWidth(width) F(length) PointGC();
    else produce EndGC() f(length) StartGC();

137
HRNSporangium\(\text{age}, \text{length}, n, \text{vigour}\)\:
\{
\text{float lengthA} = \text{max}(\text{length}, 0.02f);
\text{produce } F(\text{lengthA} \ast 2) \text{ SetWidth}(0.1) F(\text{lengthA}) \text{ SetWidth}(0.08) F(0.0) \text{ SetWidth}(0.06) F(-\text{lengthA}) \text{ SetWidth}(0.0) F(-\text{lengthA}) \text{ EndGC}();
\}

HRNSporangiumNode\(\text{age}, \text{length}, \text{vigour}, \text{width}\) > GetPos\((x,y,z)\): \{
\text{if} (\text{InsideView}(V3f(x,y,z)))
\text{produce } \text{SetColor}(\text{colourS(\text{age})}) \text{ SetWidth}(\text{width}) F(\text{length}) \text{ PointGC}();
\text{else produce } \text{EndGC}() F(\text{length}) \text{ StartGC}();
\}

HRNRhizome\(\text{age}, \text{length}, n, \text{vigour}\)\:
\{
\text{produce } \text{SetWidth}(0.04) F(\text{length}) \text{ EndGC}() \text{ Sphere}(0.02);
\}

HRNRhizomeInterior\(\text{age}, \text{length}, \text{vigour}, \text{baseWidth}, \text{widthFactor}\) > GetPos\((x,y,z)\): \{
\text{float lengthA} = \text{length} + \text{widthFactor}/5.0;
\text{if} (\text{InsideView}(V3f(x,y,z)))
\text{produce } \text{SetColor}(\text{colour(\text{age})}) \text{ SetWidth}(\text{baseWidth} \ast \text{widthFactor}) F(\text{lengthA}) \text{ PointGC}();
\text{else produce } \text{EndGC}() F(\text{lengthA}) \text{ StartGC}();
\}

HRNBase\(\text{age}, \text{pos}\)\:
\{
//\text{produce } \text{MoveTo3f}(\text{pos}); // Speed comparison with lod off
\text{produce } \text{MoveTo3f}(\text{pos}) \text{ SetWidth}(0.01) \text{ ContourSides}(\text{lod}(\text{DistanceToCamera(\text{pos})}) \ast 3);
\}
A.2 Aglaophyton

```c
#include <algorithm>
#include <cstdio>
#include <string>
#include <vector>
#include <iostream>
#include "lpfgall.h"
using namespace std;

#define nran(mean, sd)
    ((sqrt(-2*log(ran(1)))*cos(6.283185307*ran(1)))*sd+mean) // Normally distributed random number with given mean and sd
#define ranb(low,high) (ran((high)-(low))+(low))
#define sign(x) ((x > 0.0) - (x < 0.0))

#define WTSZ (80) // Water size
#define WTHT (1.5) // Water height
#define RNGT (75) // Range on terrain where germinator can appear; max 75 to stay on terrain at scale 5
#define DEATH (40) // Base age at which the plant dies
#define DFAC (25*cameraPos.scale) // Distance factor for lod
#define DELT (0.08) // Non-Varying Timestep

#define AGL_NNAG (0.24) // Age to generate a new node together with other conditions
#define AGL_SPAG (0.16) // Age to generate sporangium
#define AGL_BRAG (12) // Age to branch
#define AGL_BANG nran(38,8) // Branching Angle
#define AGL_DANG nran(0,8) // Deflection Angle
#define AGL_RROT nran(137.5/AGL_BRAG, 5) // Roll Branches
```
#define AGL_SANG nran(25,10)  // AGLSporangium pair branching angle

#define AGL_IVIG nran(0.45,0.025)  // Initial vigour value
#define AGL_RIVG nran(0.7,0.1)  // Initial vigour value (rhizome)
#define AGL_RDEF nran(0,6)  // Rhizome deflection
#define AGL_SLEN (2)  // AGLSensor Length
#define AGL_SPLN (1.0)  // AGLSporangium Length
#define AGL_MVIG (0.1)  // Vigour level at which the plant stops growing
#define AGL_LFAC (1.5)  // Length factor
#define AGL_IIWD (0.075)  // Initial internode width
#define AGL_IRIW (0.05)  // Initial rhizome internode width

#define dt nran(DELT,DELT/2.0)  // Varying Timestep

#define growthRate(age) (func(growthRateF,age)*dt*nran(0.65,0.15))
#define colour(age) floor(36+71*func(colourF,(age/57.968)))
#define vigourDecay(yh) ((1.0-((1.0-pow(func(vigourDecayF,ran(1))*(pow(yh,0.1f)),0.05f))*dt*25.0))
#define shrivel(age) func(shrivelF,(age-DEATH)/20.0)
#define sporangiumWidth(n) (func(sporangiumWidthF,n)*5.0)
#define sporangiumSplit(n,age) ((pow(abs(func(sporangiumSplitF,age/20.0f)*((12.0f-(n*min(age,75.0f)/20.0f))*0.75f)))
#define lod(dist) (6-floor(6*func(lodF,(dist/DFAC))))

VisibilityMode terrainVisibilityMode;
CameraPosition cameraPos;
float fov;
int steps;

enum ApexState { Growing, Reacting, Rebounding };
module Yardstick(int);
module YardstickI(int);
module Water();
module AGLGerminator(int);
    // Aglaophyton Germinator (Delay)
module AGLRhizomeApex(float, float, int, float, ApexState, bool);
    // Aglaophyton Rhizome Apex (Age, Length, NumberOfNodes, Vigour, State, CanBranch)
module AGLApex(float, float, int, float);
    // Aglaophyton Apex (Age, Length, NumberOfNodes, Vigour)
module AGLInternode(float, float, float, float, float);
    // Aglaophyton Internode (BaseAge, Age, Length, Vigour, Width)
module AGLRhizomeInternode(float, float, float, float, float);
    // Aglaophyton Rhizome Internode (BaseAge, Age, Length, Vigour, Width)
module AGLSporangiumGroup();
    // Aglaophyton Sporangium Group
module AGLSporangiumStalk(float, int);
    // Aglaophyton Sporangium Stalk (Segment Length, Segments)
module AGLSporangium();
    // Aglaophyton Sporangium
module AGLHalfSporangium(float, float, float);
    // Aglaophyton Half Sporangium (Age, Length, BendFactor)
module AGLSpores(float);
    // Aglaophyton Spores (Age)
module AGLStop();
    // Aglaophyton Stop sensor for contact with ground
module AGLSensor(bool);
    // Aglaophyton Sensor (Hit)
module AGLBend(float, float);
    // Aglaophyton Bend (Age, Angle)
module AGLRhizomeBend(float, float);
    // Aglaophyton RhizomeBend (Age, Angle)
module AGLBase(float, V3f);
    // Aglaophyton Base (Age, Position)

bool InsideView(V3f point) {
    V3f camDirReverse = point - cameraPos.position;
    camDirReverse.Normalize();

    return cameraPos.head*camDirReverse > abs(cos(fov/1.8));
}

float DistanceToCamera(V3f point) {
    return sqrt(pow(point.x - cameraPos.position.x, 2.0f) + pow(point.y - cameraPos.position.y, 2.0f) +
               pow(point.z - cameraPos.position.z, 2.0f));
}

Start: {
    cameraPos = GetCameraPosition(0);
    terrainVisibilityMode = Shaded;
    terrainVisibilityAll(terrainVisibilityMode);
    Backward();
    steps = 0;
}

StartEach: {
    cameraPos = GetCameraPosition(0);
    fov = 2.0 * atan(1.0/cameraPos.scale * tan(M_PI/4.0));
    steps ++;
}

derivation length: 500;
Axiom: /* SB() Yardstick(0) EB() */ CurrentTexture(2)
SB() AGLGerminator(3) EB()
SetColor(12) Terrain(cameraPos) Water();

production:

Water(): {
produce SetColor(5)
SP() MoveTo(-WTSZ,WTHT,-WTSZ) PP() MoveTo(WTSZ,WTHT,-WTSZ) PP()
MoveTo(WTSZ,WTHT,WTSZ) PP() MoveTo(-WTSZ,WTHT,WTSZ) PP() EP();
}

// A ruler for measuring plant height (and width)
Yardstick(c): {
if (c == 0) {
V3f pointOnTerrain;
V3f initPoint = V3f(-5,0,-5);

bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
if (hitTerrain) {
Printf("Yardstick Point: %.3f %.3f

%f", pointOnTerrain.x, pointOnTerrain.y, pointOnTerrain.z);
produce MoveTo3f(pointOnTerrain) Yardstick(1);
} else {
Printf("Point of FAILURE: %.3f %.3f

%f", initPoint.x, initPoint.y, initPoint.z);
Printf("Someone dropped the yardstick off a cliff\n");
produce ;
}
} else if (c <= 20) {
produce YardstickI(c) Yardstick(c+1);
}
}
AGLGerminator(delay): {
    if (delay > 0) produce AGLGerminator(delay-1);

    V3f pointOnTerrain;
    V3f initPoint = V3f(ranb(-RNGT,RNGT),0,ranb(-RNGT,RNGT));

    //initPoint = V3f(0,0,0); // TEST

    bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
    pointOnTerrain.y -= 0.5;
    if (pointOnTerrain.y < WTHT) {
        Printf("Aglaophyton spore drowned at: %f %f %f
", initPoint.x, initPoint.y, initPoint.z);
        return;
    }
    if (hitTerrain) {
        float vigour = AGL_RIVG;
        Printf("Aglaophyton germinated at: %f %f %f Vigour: %f (%i)
", pointOnTerrain.x, pointOnTerrain.y, pointOnTerrain.z, vigour, steps);
        produce RollR(ran(360)) AGLBase(0, pointOnTerrain) StartGC()
            Down(nran(60,10)) AGLRhizomeApex(0,0,0,vigour,Growing,false)
            GetPos(0,0,0) GetHead(0,0,0);
    } else {
        Printf("Point of FALIURE: %f %f %f
", initPoint.x, initPoint.y, initPoint.z);
        Printf("The spore was falling to the ground, but then it saw something so
           surprising that it missed\n");
        produce SetColor(23) MoveTo3f(initPoint) Sphere(1) F(3) Sphere(1) F(3)
            Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3)
            Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1);
            //Test
    }
}
consider: GetHead GetPos;

AGLRhizomeApex(age, length, n, vigour, state, canBranch) > GetPos(x,y,z)
    GetHead(xh,yh,zh): {
        V3f pointOnTerrain = V3f(x, max(y-10,-20.0f),z);
        terrainHeightAt(V3f(x,y,z), pointOnTerrain);

        if (vigour < AGL_MVIG) produce EndGC();

        if (canBranch && state == Growing) {
            canBranch = false;
            if (vigour < 0.15) { // Don’t let’s make a thing not grow
                // printf("I’m a go straigt to the top!\n");
                produce Elasticity(0.2) Up(30) RollR(nran(0,20))
                    AGLApex(0,0,0,vigour/1.5+AGL_IVIG) GetHead(0,0,0) GetPos(0,0,0);
            } else nproduce SB() Elasticity(0.2) Up(30) RollR(nran(0,20))
                AGLApex(0,0,0,vigour/1.5+AGL_IVIG) GetHead(0,0,0) GetPos(0,0,0) EB();
        }

        if (state == Reacting) {
            produce AGLRhizomeBend(0,nran(36,8)\*(0.5-yh*0.5)) Right(AGL_RDEF)
            AGLRhizomeInternode(0,0,0,1,AGL_IRIW) AGLSensor(false) GetPos(0,0,0)
                AGLRhizomeApex(0,0,0,vigour,Rebounding,true);
        }

        if (n >= AGL_BRAG*0.75 && vigour > 0.25 && ran(1) < 0.1) {
            produce AGLRhizomeInternode(0,0,0,1,0.05) AGLSensor(false) GetPos(0,0,0)
                Down(((nran(8,8))\*(0.9-pow(yh,2.0f))) Right(AGL_RDEF)
                SB() Right(AGL_BANG) AGLRhizomeApex(0,0,0,vigour,Growing,canBranch)
                GetPos(0,0,0) GetHead(0,0,0) EB();

        }
SB() Left(AGL_BANG) AGLRhizomeApex(0,0,0,vigour,Growing,canBranch) GetPos(0,0,0) GetHead(0,0,0) EB();

else if (age >= AGL_NNAG) {
if (y + 0.25 < pointOnTerrain.y) {
    y + 0.25 * pow(yh, 2.0f) * pow(yf, 2.0f) TEMP OLD
produce AGLRhizomeBend(0, nran(30, 8) *(0.5 - yh*0.5)) Right(AGL_RDEF)
AGLRhizomeInternode(0,0,0,1,AGL_IRIW) AGLSensor(false) GetPos(0,0,0)
AGLRhizomeApex(0,0,n+1,vigour,Reacting,false);
} else if (state == Rebounding && yh > 0.5) {
produce AGLRhizomeBend(0, nran(13, 4)) Right(AGL_RDEF)
AGLRhizomeInternode(0,0,0,1,AGL_IRIW) AGLSensor(false) GetPos(0,0,0)
AGLRhizomeApex(0,0,n+1,vigour,Growing,canBranch);
} else if (state == Rebounding) {
produce AGLRhizomeBend(0, nran(13, 4)) Right(AGL_RDEF)
AGLRhizomeInternode(0,0,0,1,AGL_IRIW) AGLSensor(false) GetPos(0,0,0)
AGLRhizomeApex(0,0,n+1,vigour,Rebounding,true);
} else {
produce AGLRhizomeInternode(0,0,0,1,AGL_IRIW) AGLSensor(false)
GetPos(0,0,0) AGLRhizomeBend(0,-(nran(11,10))*(0.9-pow(yh,2.0f)))
AGLSensor(false) GetPos(0,0,0)
Right(AGL_RDEF) AGLRhizomeApex(0,0,n+1,vigour,Growing,canBranch);
}
else produce AGLRhizomeApex(age+dt,
length+growthRate(age)*min(vigour,1.0f), n, vigour*vigourDecay(1), state, canBranch);
}

// Pipe model (modified to pow 5)
consider: AGLRhizomeInternode AGLInternode;

AGLRhizomeInternode(baseAge, age, length, vigour, width): {
float deltaT = dt;
consider: GetHead AGLBase;

AGLBase(baseAge, pos) < AGLApex(age, length, n, vigour) >

GetHead(xh,yh,zh): {
if (vigour < 0.25) return;
if (vigour < 0.35 & vigour > 0.3 & age >= AGL_SPAG & ran(1) < 0.15) {
produce AGLSporangiumGroup();
}
else if (n >= AGL_BRAG & vigour > 0.35 & ran(1) < 0.25) { // TEMP TEST TOSH
produce AGLInternode(baseAge,age,length,vigour,AGL_IIWD) GetPos(0,0,0)
SB() Left(AGL_BANG) AGLApex(0,0,0,vigour) GetHead(0,0,0) EB()
SB() Right(AGL_BANG) AGLApex(0,0,0,vigour) GetHead(0,0,0) EB();
} else if (age >= AGL_NNAG) {
produce AGLInternode(baseAge,age,length,vigour,AGL_IIWD) GetPos(0,0,0)
         RollR(AGL_RROT) AGLBend(baseAge, AGL_DANG) AGLApex(0,0,n+1,vigour);
}
produce AGLApex(age+dt, length+growthRate(age)*min(vigour,1.0f), n,
         vigour*vigourDecay(yh));
}

// Pipe model
consider: AGLInternode;

AGLInternode(baseAge, age, length, vigour, width): {
float deltaT = dt;
float i0, i1, i2, i3, i4, i5, i6, i7, widthRL, widthRR, widthR;
if (InNewRightContext(SB() AGLInternode (i0, i1, i2, i3, widthRL) EB()
        SB() AGLInternode (i4, i5, i6, i7, widthRR) EB())) {
produce AGLInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f)*AGL_LFAC, vigour,
        sqrt(pow(widthRL,2.0f)+pow(widthRR,2.0f)));}
else if (InNewRightContext(AGLInternode (i0, i1, i2, i3, widthR))) {
produce AGLInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f)*AGL_LFAC, vigour, widthR*1.02);}
else {
produce AGLInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f)*AGL_LFAC, vigour, width);
}
}

AGLSporangiumGroup(): {
float random = ran(1);
if (random < 0.85) {
produce AGLSporangiumStalk (0.1, 1) GetPos (0, 0, 0) SB() Left (AGL_SANG)
    AGLSporangiumStalk (0.1, 4) GetPos (0, 0, 0) RollR (nran (90, 10))
    AGLSporangium () EB ()
SB() Right (AGL_SANG) AGLSporangiumStalk (0.1, 4) GetPos (0, 0, 0) PointGC()
    RollR (nran (90, 10)) AGLSporangium () EB ()
} else {
    produce AGLSporangiumStalk (0.1, 2) GetPos (0, 0, 0) RollR (ran (360))
    AGLSporangium ()
}

AGLSporangium(): {
    produce AGLSpores (0) GetPos (0, 0, 0) SetColor (6) Elasticity (-0.05) EndGC ()
    CurrentContour (halfSporangium)
    SB() StartGC () AGLHalfSporangium (0, nran (1, 0.02), nran (1, 0.25))
    GetPos (0, 0, 0) EndGC () EB ()
    SB () RollR (180) StartGC () AGLHalfSporangium (0, nran (1, 0.02),
    nran (1, 0.25)) GetPos (0, 0, 0) EndGC () EB ()
}

AGLHalfSporangium (age, length, bendFactor): {
    produce AGLHalfSporangium (age+dt, length, bendFactor);
}

AGLSpores (age): {
    produce AGLSpores (age+dt)
}

consider: AGLStop;

AGLBend (age, ang): {
    if (InRightContext (AGLStop ())) produce AGBend (age+dt, ang);
else if (InRightContext(SB() AGLStop() EB()) || InRightContext(SB() EB() SB() AGLStop() EB()))
produce AGLBend(age+dt, ang) AGLStop();

produce AGLBend(age+dt, ang);
}

consider: AGLSensor;

AGLRhizomeBend(age, ang): {
bool hit;
if (InRightContext(AGLSensor(hit)) && hit) produce AGLRhizomeBend(age+dt, ang);
else produce AGLRhizomeBend(age+dt, ang-0.0125);
}

consider: GetPos AGLSensor;

AGLSensor(hit) > GetPos(x,y,z): {
bool hit2, hit3, hit4;
float i0,i1,i2;
if (hit) produce AGLSensor(true);
if (InNewRightContext(GetPos(i0,i1,i2) AGLSensor(hit2))) {
if (hit2) hit = true;
}
if (InNewRightContext(GetPos(i0,i1,i2) SB() AGLSensor(hit3) EB() SB() AGLSensor(hit4) EB())) {
if (hit3 || hit4) hit = true;
}

V3f pointOnTerrain = V3f(x, max(y-10, -20.0f), z);
terrainHeightAt(V3f(x,y,z), pointOnTerrain);
if (y - 0.05 < pointOnTerrain.y) produce AGLSensor(true);
else produce AGLSensor(hit);
}

AGLBase(age, pos): {
produce AGLBase(age + dt, pos);
}

interpretation:
maximum depth: 24;

consider: GetPos;

YardstickI(c): {
if (c % 10 == 0) produce SetWidth(1) SetColor(15) F(1);
else produce SetWidth(1) SetColor(22 + c % 2) F(1);
}

AGLRhizomeApex(age, length, vigour, n, state, canBranch): {
produce SetWidth(0) f(AGL_SLEN) EndGC();
}

AGLApex(age, length, n, vigour): {
produce SetWidth(0.04) F(0.1) EndGC() Sphere(0.02);
}

AGLInternode(baseAge, age, length, vigour, width) > GetPos(x, y, z): {
if (InsideView(V3f(x, y, z)))
produce SetColor(colour(age)) SetWidth(width) F(length) PointGC();
else produce EndGC() f(length) StartGC();
}
AGLRhizomeInternode(baseAge, age, length, vigour, width) > GetPos(x,y,z):
    {
    if (InsideView(V3f(x,y,z)))
produce SetColor(colour(age)) SetWidth(width) F(length) PointGC();
else produce EndGC() f(length) StartGC();
    }

AGLSporangiumStalk(segmentLength, numSegments) > GetPos(x,y,z): {
    if (InsideView(V3f(x,y,z)))
for (int n = 0; n < numSegments; n++)
    nproduce F(segmentLength) PointGC();
    produce ;
}

AGLHalfSporangium(age, length, bendFactor) > GetPos(x,y,z): {
    if (InsideView(V3f(x,y,z))) {
    int numSegments = lod(DistanceToCamera(V3f(x,y,z)))\*4;
    float sFac = 24/(float)numSegments;
    float sFl = numSegments;
    float sHa = numSegments/2.0;
for (int n = 0; n < numSegments ; n++)
    nproduce
        SetWidth(max(min(age,8.0f)/8.0f,0.4f)\*AGL_IWID*sporangiumWidth(n/sFl))
        Right(bendFactor*sporangiumSplit(n*sFac,age))
        F(max(min(age,8.0f)/8.0f,0.2f)*length*AGL_SPLN*((sFl+1-n)/sHa)/sFl);
    produce ;
} else produce ;
}

AGLSpores(age) > GetPos(x,y,z): {
if (InsideView(V3f(x,y,z))) {

int numSpores = pow(lod(DistanceToCamera(V3f(x,y,z))),2.0f)+4;
if (age > 10) {
for (int i = 0; i < numSpores; i++)
    nproduce SB() SetWidth(AGL_IIWD/6.0) SetColor(30) F(0.125) RollR(137.5*i)
    Right((i/(float)numSpores)*65+15) F(0.06*(i/(float)numSpores)+0.03)
    Sphere(AGL_IIWD/5.0) EB();
    produce;
}
}

AGLBend(age, ang): {
    produce Up(ang);
}

AGLRhizomeBend(age, ang): {
    produce Up(ang);
}

AGLBase(age, pos): {
    //produce MoveTo3f(pos); // Speed comparison with lod off
    produce MoveTo3f(pos) SetWidth(0.01)
    ContourSides(lod(DistanceToCamera(pos))*3);
}

A.3  *Rhynia*

#include <algorithm>
#include <cstdio>
#include <string>
#include <vector>
#include <iostream>
# include "lpfgall.h"
using namespace std;

#define nran(mean, sd)
    ((sqrt(-2*log(ran(1)))*cos(6.283185307*ran(1)))*sd+mean) // Normally
distributed random number with given mean and sd
#define ranb(low, high) (ran((high)-(low))+(low))

#define WTSZ (80) // Water size
#define WTHT (1.5) // Water height
#define RNGT (75) // Range on terrain where
    germinator can appear; max 75 to stay on terrain at scale 5
#define DEATH (40) // RHYBase age at which the plant
dies
#define DFAC (25*cameraPos.scale) // Distance factor for lod
#define DELT (0.08) // Non-Varying Timestep

#define RHY_NNAG (0.24) // Age to generate a new node
together with other conditions
#define RHY_SPAG (0.16) // Age to generate sporangium
#define RHY_BRAG (12) // Age to branch
#define RHY_BANG nran(13,2.5) // Branching Angle
#define RHY_RBAN nran(26,8) // Branching Angle (rhizome)
#define RHY_DANG nran(0,5) // Deflection Angle
#define RHY_RROT nran(137.5/RHY_BRAG,5) // Roll Branches
#define RHY_IVIG nran(0.4,0.02) // Initial vigour value
#define RHY_RIVG nran(0.7,0.1) // Initial vigour value (rhizome)
#define RHY_RDEF nran(0,4) // Rhizome deflection
#define RHY_SLEN (1) // RHYSensor Length
#define RHY_MVIG (0.1) // Vigour level at which the
    plant stops growing
#define RHY_LFAC (2.0) // Length factor
# define dt nran(DELT,DELT/2.0)     // Varying Timestep

#define growthRate(age) (func(growthRateF,age)*dt*nran(0.65,0.15))
#define colour(a) floor(36+71*func(colourF,(a/57.968)))
#define vigourDecay(yh)
    (1.0-((1.0-pow(func(vigourDecayF,ran(1))*(pow(yh,0.1f)),0.05f))*dt*25.0))
#define shrivel(age) func(shrivelF,(age-DEATH)/20.0)
#define lod(dist) (6-floor(6*func(lodF,(dist/DFAC))))

VisibilityMode terrainVisibilityMode;
CameraPosition cameraPos;
float fov;
int steps;

module Yardstick(int);
module YardstickI(int);
module Water();
module RHYGerminator(int);     // Rhynia
    Germinator (Delay)
module RHYRhizomeApex(float, float, int, float, int);     // Rhynia
    Rhizome Apex (Age, Length, NumberOfNodes, Vigour, CanBranch)
module RHYApex(float, float, int, float, bool);     // Rhynia
    Apex (Age, Length, NumberOfNodes, Vigour, Fertile)
module RHYInternode(float, float, float, float, float, bool);     // Rhynia
    Internode (BaseAge, Age, Length, Vigour, Width, Fertile)
module RHYRhizomeInternode(float, float, float, float, float);     // Rhynia
    Rhizome Internode (BaseAge, Age, Length, Vigour, Width)
module RHYSporangium();     // Rhynia
    Sporangium
module RHYStop();     // Doesn’t need to be species specific?
module RHYSensor(bool); // Sensor
   (Hit) // Doesn’t need to be species specific?
module RHYBend(float, float); // Bend
   (Age, Angle) // Doesn’t need to be species specific?
module RHYRhizomeBend(float, float); // RhizomeBend (Age, Angle) // Doesn’t need to be species specific?
module RHYBase(float, V3f); // Rhynia Base (Age, Position) // Doesn’t need to be species specific?

bool InsideView(V3f point) {
   // return true; // Speed comparison with InsideView off
   V3f camDirReverse = point-cameraPos.position;
   camDirReverse.Normalize();

   return cameraPos.head*camDirReverse > abs(cos(fov/1.8));
}

float DistanceToCamera(V3f point) {
   return sqrt(pow(point.x-cameraPos.position.x,2.0f)+pow(point.y-cameraPos.position.y,2.0f)+
}

Start: {
   cameraPos = GetCameraPosition(0);
   terrainVisibilityMode = Shaded;
   terrainVisibilityAll(terrainVisibilityMode);
   Backward();
   steps = 0;
}

StartEach: {
cameraPos = GetCameraPosition(0);
foil = 2.0 * atan(1.0/cameraPos.scale * tan(M_PI/4.0));
steps++;
}

derivation length: 550;
Axiom: /* SB() Yardstick(0) EB()*/ CurrentTexture(2)
SB() RHYGerminator(3) EB()
SetColor(12) Terrain(cameraPos) Water();

production:

Water(): {
produce SetColor(5)
SP() moveTo(-WTSZ,WTHT,-WTSZ) PP() moveTo(WTSZ,WTHT,-WTSZ) PP()
moveTo(WTSZ,WTHT,WTSZ) PP() moveTo(-WTSZ,WTHT,WTSZ) PP() EP();
}

// A ruler for measuring plant height (and width)
Yardstick(c): {
if (c == 0) {
V3f pointOnTerrain;
V3f initPoint = V3f(-5,0,-5);

bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
if (hitTerrain) {
Printf("Yardstick Point: %.3f %.3f \\
%.3f\n", pointOnTerrain.x, pointOnTerrain.y, pointOnTerrain.z);
produce moveTo3f(pointOnTerrain) Yardstick(1);
} else {
Printf("Point of FALIURE: %.3f %.3f \\
%.3f\n", initPoint.x, initPoint.y, initPoint.z);
}
Printf(“Someone dropped the yardstick off a cliff
”);
produce ;
}
} else if (c <= 20) {
produce YardstickI(c) Yardstick(c+1);
}
}

RHYGerminator(delay): {
if (delay > 0) produce RHYGerminator(delay-1);

V3f pointOnTerrain;
V3f initPoint = V3f(ranb(-RNGT,RNGT),0,ranb(-RNGT,RNGT));

initPoint = V3f(0,0,0); // TEST

bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
pointOnTerrain.y -= 0.5;
if (pointOnTerrain.y < WTHT) {
Printf(“Rhynia spore drowned at: %f %f
%f
”,initPoint.x,initPoint.y,initPoint.z);
return;
}
if (hitTerrain) {
Printf(“Rhynia germinated at: %f %f %f
(%i)
”,pointOnTerrain.x,pointOnTerrain.y,pointOnTerrain.z,steps);
produce RollR(ran(360)) RHYBase(0,pointOnTerrain) StartGC()
   Down(nran(60,10)) RHYRhizomeApex(0,0,0,RHY_RIVG,0) GetPos(0,0,0)
   GetHead(0,0,0);
} else {
Printf(“Point of FALIURE: %f %f
%f
”,initPoint.x,initPoint.y,initPoint.z);

158
Printf("The spore was falling to the ground, but then it saw something so surprising that it missed\n");
produce SetColor(23) MoveTo3f(initPoint) Sphere(1) F(3) Sphere(1) F(3)
    Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3)
    Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1);
//Test
}
}

consider: GetHead GetPos;

RHYRhizomeApex(age, length, n, vigour, canBranch) > GetPos(x,y,z)
    GetHead(xh,yh,zh): {
V3f pointOnTerrain = V3f(x,max(y-10,-20.0f),z);
terrainHeightAt(V3f(x,y,z), pointOnTerrain);
if (vigour < RHY_MVIG) produce EndGC();

if (canBranch > 125) {
    canBranch = 0;
    if (vigour < 0.15) { // Don’t let’s make a thing not grow
        //printf("I’m a go straitg to the top!\n");
        produce Up(25.0*(1-yh)) RHYInternode(age,0,0,(vigour/2.0)+0.5,0.05,false)
            GetPos(0,0,0) Up(25.0*(1-yh))
            RHYInternode(age,0,0,(vigour/2.0)+0.5,0.05,false) GetPos(0,0,0)
            Up(25.0*(1-yh))
        Elasticity(0.05) RHYApex(0,0,0,vigour/1.5+RHY_IVIG,false) GetHead(0,0,0);
    } else nproduce SB() Up(25.0*(1-yh))
        RHYInternode(age,0,0,vigour+0.1,0.05,false) GetPos(0,0,0)
        Up(25.0*(1-yh)) RHYInternode(age,0,0,vigour+0.1,0.05,false)
        GetPos(0,0,0) Up(25.0*(1-yh))
        Elasticity(0.05) RHYApex(0,0,0,vigour/1.5+RHY_IVIG,false) GetHead(0,0,0)
if (n >= RHY_BRAG * 1.5 && vigour > 0.25 && ran(1) < 0.5) {
    produce RHYRhizomeInternode(0,0,0,1,0.05) RHYSensor(false) GetPos(0,0,0)
    Down((nran(15,8)) * (0.9 - pow(yh, 2.0f))) Right(RHY_RDEF)
    SB() Right(RHY_RBAN) RHYRhizomeApex(0,0,0,vigour,canBranch) GetPos(0,0,0)
    GetHead(0,0,0) EB()
    SB() Left(RHY_RBAN) RHYRhizomeApex(0,0,0,vigour,canBranch) GetPos(0,0,0)
    GetHead(0,0,0) EB();
} else if (age >= RHY_NNAG) {
    if (y - 0.1 < pointOnTerrain.y) {
        produce RHYRhizomeBend(0, nran(30,3) * (0.5 - yh * 0.5)) Right(RHY_RDEF)
        RHYRhizomeInternode(0,0,0,1,0.05) RHYSensor(false) GetPos(0,0,0)
        RHYRhizomeApex(0,0,n+1,vigour,canBranch);
    } else {
        produce RHYRhizomeInternode(0,0,0,1,0.05) RHYSensor(false) GetPos(0,0,0)
        RHYRhizomeBend(0, -(nran(15,8)) * (0.9 - pow(yh, 2.0f))) RHYSensor(false)
        GetPos(0,0,0)
        Right(RHY_RDEF) RHYRhizomeApex(0,0,n+1,vigour,canBranch);
    }
} else produce RHYRhizomeApex(age+dt,
    length+growthRate(age)*min(vigour,1.0f), n, vigour*vigourDecay(1),
    canBranch+(nran(2,1)));
}

// Pipe model (modified to pow 5)
consider: RHYRhizomeInternode RHYInternode;

RHYRhizomeInternode(baseAge, age, length, vigour, width): {
    float deltaT = dt;
    float i0, i1, i2, i3, i4, i5, i6, i7, widthRL, widthRR, widthR;

160
bool fert;
if (InNewRightContext(SB() RHYRhizomeInternode(i0, i1, i2, i3, widthRL)
    EB() SB() RHYRhizomeInternode(i4, i5, i6, i7, widthRR) EB())) {
    produce RHYRhizomeInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f)*0.5, vigour,
        pow(pow(widthRL,5.0f)+pow(widthRR,5.0f),0.2f));
} else if (InNewRightContext(SB() RHYInternode(i0, i1, i2, i3, widthRL,
    fert) EB() RHYRhizomeInternode(i4, i5, i6, i7, widthRR))) {
    produce RHYRhizomeInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f)*0.5, vigour,
        pow(pow(widthRL,5.0f)+pow(widthRR,5.0f),0.2f));
} else if (InNewRightContext(RHYRhizomeInternode(i0, i1, i2, i3, widthR))
    || InNewRightContext(RHYInternode(i0, i1, i2, i3, widthR, fert))) {
    produce RHYRhizomeInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f)*0.5, vigour, widthR*1.01);
} else {
    produce RHYRhizomeInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f)*0.5, vigour, width);}
}

call: GetHead RHYBase;

RHYBase(baseAge, pos) < RHYApex(age, length, n, vigour, fertile) >
    GetHead(xh,yh,zh) { 
    if (vigour < 0.2) return;
    if (vigour < 0.3 && vigour > 0.25 && age >= RHY_SPAG && ran(1) < 0.2 &&
        fertile) {
        produce RHYSporangium() GetPos(0,0,0);
    } else if (n >= RHY_BRAG && vigour > 0.35 && ran(1) < 0.25) { // TEMP
        TEST TOS
        produce RHYInternode(baseAge,age,length,vigour,0.05,false) GetPos(0,0,0)
SB() Left(RHY_BANG) RHYApex(0,0,0,vigour,fertile) GetHead(0,0,0) EB()
SB() Right(RHY_BANG) RHYApex(0,0,0,vigour,fertile) GetHead(0,0,0) EB();
else if (age >= RHY_NNAG) {
    produce RHYInternode(baseAge,age,length,vigour,0.05,true) GetPos(0,0,0)
    RollR(RHY_RROT) RHYBend(baseAge, RHY_DANG)
    RHYApex(0,0,n+1,vigour,fertile);
}
produce RHYApex(age+dt, length+growthRate(age)*min(vigour,1.0f), n,
    vigour*vigourDecay(yh), fertile);

// Pipe model
consider: RHYInternode;

RHYInternode(baseAge, age, length, vigour, width, fertile): {
  float deltaT = dt;

  if (ran(1) < 0.05 && fertile) {
    produce RHYInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f)*RHY_LFAC, vigour, width, false)
    SB() Left(RHY_BANG*2.5) RollR(RHY_RROT) Elasticity(0.08)
    RHYApex(age+deltaT, 0, 0, vigour*0.85, true) GetHead(0,0,0) EB();
  }

  float i0, i1, i2, i3, i4, i5, i6, i7, widthRL, widthRR, widthR;
  bool fert;
  if (InNewRightContext(SB()) RHYInternode (i0, i1, i2, i3, widthRL, fert)
      EB() SB() RHYInternode (i4, i5, i6, i7, widthRR, fert) EB())) {
    produce RHYInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f)*RHY_LFAC, vigour,
        pow(pow(widthRL,2.0f)+pow(widthRR,2.0f),0.5f), false);
  } else if (InNewRightContext(RHYInternode (i0, i1, i2, i3, widthRL, fert)
SB() RHYInternode(i4, i5, i6, i7, widthRR, fert) EB() {  
produce RHYInternode(baseAge+deltaT, age+deltaT,  
length+growthRate(age)*min(vigour,1.0f)*RHY_LFAC, vigour,  
pow(pow(widthRL,2.0f)+pow(widthRR,2.0f),0.5f), false);  
}  
else if (InNewRightContext(RHYInternode(i0, i1, i2, i3, widthR, fert))) {  
produce RHYInternode(baseAge+deltaT, age+deltaT,  
length+growthRate(age)*min(vigour,1.0f)*RHY_LFAC, vigour, widthR*1.01, false);  
}  
else {  
produce RHYInternode(baseAge+deltaT, age+deltaT,  
length+growthRate(age)*min(vigour,1.0f)*RHY_LFAC, vigour, width, false);  
}  
}  
}  

consider: RHYStop;  
RHYBend(age, ang): {  
if (InRightContext(RHYStop())) produce RHYBend(age+dt, ang);  
else if (InRightContext(SB() RHYStop() EB()) || InRightContext(SB() EB())  
SB() RHYStop() EB()))  
produce RHYBend(age+dt, ang) RHYStop();  
}  
produce RHYBend(age+dt, ang);  
}  
}  

consider: RHYSensor;  
RHHRhizomeBend(age, ang): {  
bool hit;  
if (InRightContext(RHYSensor(hit)) && hit) produce RHHRhizomeBend(age+dt,
ang);
else produce RHYRhizomeBend(age+dt, ang-0.0125);
}

consider: GetPos RHYSensor;

RHYSensor(hit) > GetPos(x,y,z): {
    bool hit2, hit3, hit4;
    float i0,i1,i2;
    if (hit) produce RHYSensor(true);
    if (InNewRightContext(GetPos(i0,i1,i2) RHYSensor(hit2))) {
        if (hit2) hit = true;
    }
    if (InNewRightContext(GetPos(i0,i1,i2) SB() RHYSensor(hit3) EB() SB()
        RHYSensor(hit4) EB())) {
        if (hit3 || hit4) hit = true;
    }

    V3f pointOnTerrain = V3f(x, max(y-10,-20.0f),z);
    terrainHeightAt(V3f(x,y,z), pointOnTerrain);

    if (y-0.05 < pointOnTerrain.y) produce RHYSensor(true);
    else produce RHYSensor(hit);
}

RHYBase(age, pos): {
    produce RHYBase(age+dt, pos);
}

interpretation:
maximum depth: 2;
consider: GetPos;

YardstickI(c): {
  if (c%10 == 0) produce SetWidth(1) SetColor(15) F(1);
  else produce SetWidth(1) SetColor(22+c%2) F(1);
}

RHYRhizomeApex(age,length,vigour,n,canBranch): {
  produce SetWidth(0) f(RHY_SLEN) EndGC();
}

RHYApex(age, length, n, vigour, fertile): {
  //if (fertile) produce SetColor(14) SetWidth(0.04) F(length+2) EndGC()
  Sphere(0.02);
  produce SetWidth(0.04) F(length) EndGC() Sphere(0.02);
}

RHYInternode(baseAge, age, length, vigour, width, fertile) > GetPos(x,y,z): {
  if (InsideView(V3f(x,y,z)))
  produce SetColor(colour(age)) SetWidth(width) F(length) PointGC();
  else produce EndGC() f(length) StartGC();
}

RHYRhizomeInternode(baseAge, age, length, vigour, width) > GetPos(x,y,z): {
  if (InsideView(V3f(x,y,z)))
  produce SetColor(colour(age)) SetWidth(width) F(length) PointGC();
  else produce EndGC() f(length) StartGC();
}

RHYSporangium() > GetPos(x,y,z): {
if (InsideView(V3f(x,y,z))) {
    produce SetColor(6) Surface(0,0.075) SetWidth(0.05) PointGC() F(0.1)
    SetWidth(0.04) PointGC() Sphere(0.02);
} else produce ;
}

RHYBend(age, ang): {
    produce Up(ang);
}

RHYRhizomeBend(age, ang): {
    produce Up(ang);
}

RHYBase(age, pos): {
    //produce MoveTo3f(pos); // Speed comparison with lod off
    produce MoveTo3f(pos) SetWidth(0.01)
    ContourSides(lod(DistanceToCamera(pos))*3);
}

A.4 Nothia

#include <algorithm>
#include <cstdio>
#include <string>
#include <vector>
#include <iostream>
#include "lpfgall.h"
using namespace std;

#define nran(mean, sd)
    ((sqrt(-2*log(ran(1)))*cos(6.283185307*ran(1)))*sd+mean)  // Normally
distributed random number with given mean and sd

#define ranb(low,high) (ran((high)-(low))+(low))
#define sign(x) ((x > 0.0) - (x < 0.0))

#define WTSZ (80) // Water size
#define WTHT (1.5) // Water height
#define RNGT (75) // Range on terrain where germinator can appear; max 75 to stay on terrain at scale 5
#define DEATH (40) // Base age at which the plant dies
#define DFAC (25*cameraPos.scale) // Distance factor for lod
#define DELT (0.08) // Non-Varying Timestep

#define NTH_NNAG (0.24) // Age to generate a new node
together with other conditions // NOT USED?
#define NTH_SPAG (0.16) // Age to generate sporangium

#define NTH_BRAG (10) // Age to branch
#define NTH_BANG nran(40,5) // Branching Angle
#define NTH_RBAN nran(26,8) // Branching Angle (rhizome)
#define NTH_DANG nran(0,12) // Deflection Angle
#define NTH_RROT nran(137.5/NTH_BRAG , 5)// Roll Branches
#define NTH_SANG nran(35,5) // Sporangium angle
#define NTH_IVIG nran(0.4,0.02) // Initial vigour value
#define NTH_RIVG nran(0.6,0.1) // Initial vigour value (rhizome)
#define NTH_RDEF nran(0,4) // Rhizome deflection
#define NTH_SLEN (1) // Sensor Length
#define NTH_SPLN (0.4) // Sporangium Length
#define NTH_MVIG (0.1) // Vigour level at which the plant stops growing
#define NTH_LFAC (2.0) // Length factor

#define dt nran(DELT,DELT/2.0) // Varying Timestep
#define growthRate(age) (func(growthRateF,age)*dt\*nran(0.65,0.15))
#define colour(a) floor(36+71*func(colourF,(a/57.968)))
#define vigourDecay(yh)
    (1.0-((1.0-pow(func(vigourDecayF,ran(1)))*(pow(yh,0.1f)),0.05f))*dt*25.0))
#define shrivel(age) func(shrivelF,(age-DEATH)/20.0)
#define sporangiumWidth(n) (func(sporangiumWidthF,n)*0.2)
#define sporangiumSplit(n,age)
    ((pow(abs(func(sporangiumSplitF,age/20.0f))*((12.0f-(n\*min(age,75.0f)/20.0f))*0.75f),0.4f)*3.0*sign((12.0f-(n\*min(age,75.0f)/20.0f))))
#define lod(dist) (6-floor(6*func(lodF,(dist/DFAC))))

VisibilityMode terrainVisibilityMode;
CameraPosition cameraPos;
float fov;
int steps;

module Yardstick(int);
module YardstickI(int);
module Water();
module NTHGerminator(int);  // Nothia
    Germinator (Delay)
module NTHRhizomeApex(float, float, int, float, int);  // Nothia
    Rhizome Apex (Age, Length, NumberOfNodes, Vigour, CanBranch)
module NTHApex(float, float, int, float);  // Nothia
    Apex (Age, Length, NumberOfNodes, Vigour)
module NTHInternode(float, float, float, float, float);  // Nothia
    Internode (NTHBaseAge, Age, Length, Vigour, Width)
module NTHRhizomeInternode(float, float, float, float, float);  // Nothia
    Rhizome Internode (NTHBaseAge, Age, Length, Vigour, Width)
module NTHSporangium();  // Nothia
    Sporangium
module NTHHalfSporangium(float, float, float);  // Nothia
Half Sporangium (Age, Length, NTHBendFactor)

module NTHStop(); // Probably doesn’t need to be plant specific

module NTHSensor(bool); // Nothia
    Sensor (Hit) // May not need to be plant specific

module NTHBend(float, float); // Nothia
    Bend (Age, Angle) // May not need to be plant specific

module NTHRhizomeBend(float, float); // Nothia
    RhizomeBend (Age, Angle) // May not need to be plant specific

module NTHBase(float, V3f); // Nothia
    Base (Age, Position)

bool InsideView(V3f point) {
    //return true; // Speed comparison with InsideView off

    V3f camDirReverse = point-cameraPos.position;
    camDirReverse.Normalize();

    //printf("--------------------------------------------------\n");
    //printf("Point at (%f,%f,%f)\n", point.x, point.y, point.z);
    //printf("Direction to camera is (%f,%f,%f)\n", camDirReverse.x,
            camDirReverse.y, camDirReverse.z);
    //printf("Heading of camera is (%f,%f,%f)\n", cameraPos.head.x,
            cameraPos.head.y, cameraPos.head.z);
    //printf("Resultant angle is %f vs. a fov of %f\n",
            cameraPos.head*camDirReverse, abs(cos(fov/2.0)));

    return cameraPos.head*camDirReverse > abs(cos(fov/1.8));
}

float DistanceToCamera(V3f point) {
    return

\[
\sqrt{\text{pow}(\text{point.x-camerapos.position.x}, 2.0f) + \text{pow}(\text{point.y-camerapos.position.y}, 2.0f) + \text{pow}(\text{point.z-camerapos.position.z}, 2.0f)}
\]

Start: {
    camerapos = GetCameraPosition(0);
    terrainvisibilitymode = Shaded;
    terrainvisibilityAll(terrainvisibilitymode);
    Backward();
    steps = 0;
}

StartEach: {
    camerapos = GetCameraPosition(0);
    fov = 2.0 * \text{atan}(1.0/camerapos.scale * \text{tan}(M_PI/4.0));
    steps++;
}

derivation length: 350;

Axiom: /* SB() Yardstick(0) EB() */ CurrentTexture(2)
    CurrentContour(circle)
    SB() NTHGerminator(3) EB()
    SetColor(12) Terrain(camerapos) Water();

production:

Water(): {
    produce SetColor(5)
    SP() MoveTo(-WTSZ,WTHT,-WTSZ) PP() MoveTo(WTSZ,WTHT,-WTSZ) PP()
    MoveTo(WTSZ,WTHT,WTSZ) PP() MoveTo(-WTSZ,WTHT,WTSZ) PP() EP();
}

// A ruler for measuring plant height (and width)

170
Yardstick(c): {
    if (c == 0) {
        V3f pointOnTerrain;
        V3f initPoint = V3f(-5,0,-5);
        bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
        if (hitTerrain) {
            printf("Yardstick Point: %f %f
", pointOnTerrain.x, pointOnTerrain.y, pointOnTerrain.z);
            produce MoveTo3f(pointOnTerrain) Yardstick(1);
        } else {
            printf("Point of FAILURE: %f %f
", initPoint.x, initPoint.y, initPoint.z);
            printf("Someone dropped the yardstick off a cliff\n");
            produce;
        }
    } else if (c <= 20) {
        produce YardstickI(c) Yardstick(c+1);
    }
}

NTHGerminator(delay): {
    if (delay > 0) produce NTHGerminator(delay-1);
    V3f pointOnTerrain;
    V3f initPoint = V3f(ranb(-RNGT,RNGT),0,ranb(-RNGT,RNGT));
    //initPoint = V3f(0,0,0); // TEST
    bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
    pointOnTerrain.y -= 0.5;
    if (pointOnTerrain.y < WTHT) {
Printf("Nothia spore drowned at: %f %f
\n", initPoint.x, initPoint.y, initPoint.z);
return;
}
if (hitTerrain) {
float vigour = NTH_RIVG;
Printf("Nothia germinated at: %f %f %f Vigour: %f
\n", pointOnTerrain.x, pointOnTerrain.y, pointOnTerrain.z, vigour, steps);
produce RollR(ran(360)) NTHBase(0, pointOnTerrain) StartGC()
    Down(nran(60,10)) NTHRizomeApex(0,0,0,vigour,0) GetPos(0,0,0)
    GetHead(0,0,0);
} else {
Printf("Point of FALIURE: %f %f
\n", initPoint.x, initPoint.y, initPoint.z);
Printf("The spore was falling to the ground, but then it saw something so
surprising that it missed\n\n");
produce SetColor(23) MoveTo3f(initPoint) Sphere(1) F(3) Sphere(1) F(3)
    Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3)
Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1);  //Test
}
}

consider: GetHead GetPos;
NTHRizomeApex(age, length, n, vigour, canBranch) > GetPos(x,y,z)
    GetHead(xh,yh,zh): {
V3f pointOnTerrain = V3f(x, max(y-10,-20.0f), z);
terrainHeightAt(V3f(x,y,z), pointOnTerrain);

if (vigour < NTH_MVIG) produce EndGC();
if (canBranch > 125/(DELT*12.0)) {
    canBranch = 0;
    if (vigour < 0.15) { // Don’t let’s make a thing not grow
        // printf("I’m a go straigt to the top!\n");
        produce Elasticity(0.05) Up(25.0*(1-yh))
            NTHInternode(age,0,0,(vigour/2.0)+0.5,0.05) GetPos(0,0,0)
            Up(25.0*(1-yh)) NTHInternode(age,0,0,(vigour/2.0)+0.5,0.05)
            GetPos(0,0,0) Up(25.0*(1-yh))
        Elasticity(0.05) NTHApex(0,0,0,vigour/1.5+NTH_IVIG) GetHead(0,0,0);
    } else nproduce Elasticity(0.05) SB() Up(25.0*(1-yh))
        NTHInternode(age,0,0,vigour+0.1,0.05) Up(25.0*(1-yh)) GetPos(0,0,0)
        NTHInternode(age,0,0,vigour+0.1,0.05) GetPos(0,0,0) Up(25.0*(1-yh))
        Elasticity(0.05) NTHApex(0,0,0,vigour/1.5+NTH_IVIG) GetHead(0,0,0) EB();
}

if (n >= NTH_BRAG*1.5 && vigour > 0.25 && ran(1) < 0.1) {
    produce NTHRhizomeInternode(0,0,0,1,0.05) NTHSensor(false) GetPos(0,0,0)
        Down((nran(15,8))*(0.9-pow(yh,2.0f))) Right(NTH_RDEF)
    SB() Right(NTH_RBAN) NTHRhizomeApex(0,0,0,vigour,cBranch) GetPos(0,0,0)
    GetHead(0,0,0) EB();
    SB() Left(NTH_RBAN) NTHRhizomeApex(0,0,0,vigour,cBranch) GetPos(0,0,0)
    GetHead(0,0,0) EB();
} else if (age >= NTH_NNAG) {
    if (y+0.33 < pointOnTerrain.y) {
        produce NTHRhizomeBend(0, nran(30,3)*(0.5-yh*0.5)) Right(NTH_RDEF)
        NTHRhizomeInternode(0,0,0,1,0.05) NTHSensor(false) GetPos(0,0,0)
        NTHRhizomeApex(0,0,n+1,vigour,cBranch);
    } else {
        produce NTHRhizomeInternode(0,0,0,1,0.05) NTHSensor(false) GetPos(0,0,0)
            NTHRhizomeBend(0,-(nran(15,8))*(0.9-pow(yh,2.0f))) NTHSensor(false)
            GetPos(0,0,0)
        Right(NTH_RDEF) NTHRhizomeApex(0,0,n+1,vigour,cBranch);
    }
}
} else produce NTHRhizomeApex(age+dt,
    length+growthRate(age)*min(vigour,1.0f), n,  
    vigour*vigourDecay(1),
    canBranch+(nran(2,1)));}

// Pipe model (modified to pow 5)
consider: NTHRhizomeInternode NTHInternode;

NTHRhizomeInternode(baseAge, age, length, vigour, width): {
    float deltaT = dt;
    float i0, i1, i2, i3, i4, i5, i6, i7, widthRL, widthRR, widthR;
    bool a;
    if (InNewRightContext(SB() NTHRhizomeInternode(i0, i1, i2, i3, widthRL) 
        EB() SB() NTHRhizomeInternode(i4, i5, i6, i7, widthRR) EB())) {
        produce NTHRhizomeInternode(baseAge+deltaT, age+deltaT,
            length+growthRate(age)*min(vigour,1.0f)*0.5, vigour,
            pow(pow(widthRL,5.0f)+pow(widthRR,5.0f),0.2f));
    } else if (InNewRightContext(SB() NTHInternode(i0, i1, i2, i3, widthRL) 
        EB() NTHRhizomeInternode(i4, i5, i6, i7, widthRR))) {
        produce NTHRhizomeInternode(baseAge+deltaT, age+deltaT,
            length+growthRate(age)*min(vigour,1.0f)*0.5, vigour,
            pow(widthRL,5.0f)+pow(widthRR,5.0f),0.2f));
    } else if (InNewRightContext(NTHRhizomeInternode(i0, i1, i2, i3, widthR)) 
        || InNewRightContext(NTHInternode(i0, i1, i2, i3, widthR))) {
        produce NTHRhizomeInternode(baseAge+deltaT, age+deltaT,
            length+growthRate(age)*min(vigour,1.0f)*0.5, vigour, widthR*1.01);
    } else {
        produce NTHRhizomeInternode(baseAge+deltaT, age+deltaT,
            length+growthRate(age)*min(vigour,1.0f)*0.5, vigour, width);
    }
}
consider: GetHead NTHBase;

NTHBase(baseAge, pos) < NTHApex(age, length, n, vigour) >
  GetHead(xh,yh,zh): {
    if (vigour < 0.2) return;
    if (vigour < 0.24)
      nproduce SB() RollR(137.5*n) Right(NTH_SANG) NTHSporangium()
      GetPos(0,0,0) EB();
    if (n >= NTH_BRAG && vigour > 0.35 && ran(1) < 0.25) { // TEMP TEST TOSH
      produce NTHInternode(baseAge,age,length,vigour,0.075) GetPos(0,0,0) SB()
        Left(NTH_BANG) NTHApex(0,0,0,vigour) GetHead(0,0,0) EB()
        SB() Right(NTH_BANG) NTHApex(0,0,0,vigour) GetHead(0,0,0) EB();
    } else if (age >= NTH_NNAG) {
      produce NTHInternode(baseAge,age,length,vigour,0.075) GetPos(0,0,0)
        RollR(NTH_RROT) NTHBend(baseAge, NTH_DANG) NTHApex(0,0,n+1,vigour);
    }
    produce NTHApex(age+dt, length+growthRate(age)*min(vigour,1.0f), n,
      vigour*vigourDecay(yh));
  }

  // Pipe model
  consider: NTHInternode;

NTHInternode(baseAge, age, length, vigour, width): {
  float deltaT = dt;

  float i0, i1, i2, i3, i4, i5, i6, i7, widthRL, widthRR, widthR;
  if (InNewRightContext(SB() NTHInternode (i0, i1, i2, i3, widthRL) EB())
    SB() NTHInternode (i4, i5, i6, i7, widthRR) EB())) {
    produce NTHInternode(baseAge+deltaT, age+deltaT,
      length+growthRate(age)*min(vigour,1.0f)*NTH_LFAC, vigour,
\[
pow(pow(widthRL, 2.0f) + pow(widthRR, 2.0f), 0.5f));
\]

) else if (InNewRightContext(NTHInternode (i0, i1, i2, i3, widthRL) SB())
NTHInternode (i4, i5, i6, i7, widthRR) EB()) {
produce NTHInternode(baseAge+deltaT, age+deltaT,
length+growthRate(age)*min(vigour, 1.0f)*NTH_LFAC, vigour,
pow(pow(widthRL, 2.0f) + pow(widthRR, 2.0f), 0.5f));
) else if (InNewRightContext(NTHInternode (i0, i1, i2, i3, widthR))) {
produce NTHInternode(baseAge+deltaT, age+deltaT,
length+growthRate(age)*min(vigour, 1.0f)*NTH_LFAC, vigour,
widthR*1.005);
} else {
produce NTHInternode(baseAge+deltaT, age+deltaT,
length+growthRate(age)*min(vigour, 1.0f)*NTH_LFAC, vigour, width);
}
}

NTHSporangium(): {
/* produce SetColor(7) Elasticity(-0.1) EndGC()

CurrentContour(halfSporangium2)
SB() StartGC() NTHHalfSporangium(0, nran(1,0.00), nran(1,0.25)) EndGC()
EB()
SB() RollR(180) StartGC() NTHHalfSporangium(0, nran(1,0.00),
nran(1,0.25)) EndGC() EB(); */
produce SetColor(7) Elasticity(-0.1) EndGC() CurrentContour(circle)
SB() StartGC() NTHHalfSporangium(0, nran(1,0.02), nran(1,0.25))
GetPos(0,0,0) EndGC() EB();
/* produce SetColor(7) F(0.1) EndGC() Sphere(0.1); */
}

NTHHalfSporangium(age, length, bendFactor): {
produce NTHHalfSporangium(age+dt, length, bendFactor);
}
consider: NTHStop;

NTHBend(age, ang): {
    if (InRightContext(NTHStop())) produce NTHBend(age+dt, ang);
    else if (InRightContext(SB() NTHStop() EB()) || InRightContext(SB() EB() SB() NTHStop() EB()))
        produce NTHBend(age+dt, ang) NTHStop();

    produce NTHBend(age+dt, ang);
}

consider: NTHSensor;

NTHRhizomeBend(age, ang): {
    bool hit;
    if (InRightContext(NTHSensor(hit)) && hit) produce NTHRhizomeBend(age+dt, ang);
    else produce NTHRhizomeBend(age+dt, ang-0.0125);
}

consider: GetPos NTHSensor;

NTHSensor(hit) > GetPos(x,y,z): {
    bool hit2, hit3, hit4;
    float i0,i1,i2;
    if (hit) produce NTHSensor(true);
    if (InNewRightContext(GetPos(i0,i1,i2) NTHSensor(hit2))) {
        if (hit2) hit = true;
    }
    if (InNewRightContext(GetPos(i0,i1,i2) SB() NTHSensor(hit3) EB() SB() NTHSensor(hit4) EB())) {

if (hit3 || hit4) hit = true;
}

V3f pointOnTerrain = V3f(x, max(y-10,-20.0f),z);
terrainHeightAt(V3f(x,y,z), pointOnTerrain);

if (y - 0.05 < pointOnTerrain.y) produce NTHSensor(true);
else produce NTHSensor(hit);
}

NTHBase(age, pos): {
produce NTHBase(age+dt, pos);
}

interpretation:
maximum depth: 25;

consider: GetPos;

YardstickI(c): {
if (c %10 == 0) produce SetWidth(1) SetColor(15) F(1);
else produce SetWidth(1) SetColor(22+c%2) F(1);
}

NTHRhizomeApex(age, length, vigour, n, canBranch): {
produce SetWidth(0) f(NTH_SLEN) EndGC();
}

NTHApex(age, length, n, vigour): {
produce SetWidth(0.04) F(0.1) EndGC() Sphere(0.02);
}
// Try to make the surface less regular?
NTHInternode(baseAge, age, length, vigour, width) > GetPos(x,y,z): {
  if (InsideView(V3f(x,y,z))) {
    int lod = lod(DistanceToCamera(V3f(x,y,z)));
    if (lod >= 4) {
      produce SetColor(colour(age)) SetWidth(width) /*F(length)
        PointGC();*/
      BlendedContour(circle, ovalA, min(max(2.0 - vigour*3.0, 0.0), 1.0))
        F(length/4.0) PointGC()
      CurrentContour(circle) F(length/4.0) PointGC()
      BlendedContour(circle, ovalB, min(max(2.0 - vigour*3.0, 0.0), 1.0))
        F(length/4.0) PointGC()
      CurrentContour(circle) F(length/4.0) PointGC();
    } else if (lod >= 2) {
      produce SetColor(colour(age)) SetWidth(width)
      BlendedContour(circle, ovalA, min(max(2.0 - vigour*3.0, 0.0), 1.0))
        F(length/4.0) PointGC() F(length/4.0) PointGC()
      CurrentContour(circle) F(length/4.0) PointGC() F(length/4.0) PointGC();
    } else {
      produce SetColor(colour(age)) SetWidth(width)
      F(length/4.0) PointGC() F(length/4.0) PointGC() F(length/4.0) PointGC()
        F(length/4.0) PointGC();
    }
  } else produce EndGC() f(length/4.0) f(length/4.0) f(length/4.0)
    f(length/4.0) StartGC();
}
NTHHalfSporangium\( (\text{age}, \text{length}, \text{bendFactor}) \rightarrow \text{GetPos}(x,y,z) \): 

\[
\text{if (InsideView(V3f(x,y,z)))} \{ \\
\text{int numSegments = lod(DistanceToCamera(V3f(x,y,z)))*2;} \\
\text{float sFl = numSegments;} \\
\text{for (int n = 0; n < numSegments ; n++)} \\
\text{nproduce} \\
\text{SetWidth(max(min(\text{age},8.0f)/8.0f,0.4f)*sporangiumWidth(n/(sFl-1.0)))} \\
F(max(min(\text{age},8.0f)/8.0f,0.2f)*length*NTH_SPLN*((sFl+1-n)/sFl)/sFl); \\
\text{produce ;} \\
\} \text{ else produce ;} \\
\}
\]

\[
\text{NTHBend(\text{age}, \text{ang}) : \{} \\
\text{produce Up(\text{ang}) ;} \\
\}
\]

\[
\text{NTHRhizomeBend(\text{age}, \text{ang}) : \{} \\
\text{produce Up(\text{ang}) ;} \\
\}
\]

\[
\text{NTHBase(\text{age}, \text{pos}) : \{} \\
\text{//produce MoveTo3f(pos);} // Speed comparison with lod off} \\
\text{produce MoveTo3f(pos) SetWidth(0.01)} \\
\text{ContourSides(lod(DistanceToCamera(pos))*3);} \\
\}
\]

A.5 Ventarura

#include <algorithm>
#include <cstdio>
#include <string>
#include <vector>
#include <iostream>
#include "lpfgall.h"

using namespace std;

#define nran(mean, sd)
    ((sqrt(-2*log(ran(1)))*cos(6.283185307*ran(1)))*sd+mean) // Normally
distributed random number with given mean and sd
#define ranb(low, high) (ran((high)-(low))+(low))
#define sign(x) ((x > 0.0) - (x < 0.0))

#define WTSZ (80) // Water size
#define WTHT (1.5) // Water height
#define RNGT (75) // Range on terrain where
    germinator can appear; max 75 to stay on terrain at scale 5
#define DEATH 40 // Base age at which the plant
dies
#define DFAC (25*cameraPos.scale) // Distance factor for lod
#define DELT (0.08) // Non-Varying Timestep

#define VNT_NNAG (0.24) // Age to generate a new node
#define VNT_BRAG (5) // Age to branch
#define VNT_BANG nran(18,3) // Branching Angle
#define VNT_DANG nran(0,3) // Deflection Angle
#define VNT_RROT nran(90.0/VNT_BRAG,5) // Roll Branches
#define VNT_SANG nran(80,9) // Sporangium angle
#define VNT_IVIG nran(1.0,0.1) // Initial vigour value
#define VNT_SPLN (0.15) // Sporangium Length
#define VNT_IIWD (0.075) // Initial internode width

#define dt nran(DELT,DELT/2.0) // Varying Timestep
# define growthRate(age) (func(growthRateF, age) * dt * nran(0.65, 0.15))
# define colour(a) floor(36 + 71 * func(colourF, (a / 60.0)))
# define vigourDecay(yh)
    (1.0 - ((1.0 - pow(func(vigourDecayF, ran(1)) * (pow(yh, 0.1f)), 0.05f)) * dt * 25.0))
// #define vigourDecay(yh)
    pow(func(vigourDecayF, ran(1)) * (pow(yh, 0.125f)), 0.125f)
# define shrivel(age) func(shrivelF, (age - DEATH) / 20.0)
# define sporangiumWidth(n) (func(sporangiumWidthF, n) * 5.0)
# define sporangiumSplit(n, age)
    ((pow(abs(func(sporangiumSplitF, age / 20.0f)) * (12.0f - (n * min(age, 75.0f) / 20.0f)) * 0.75f))
# define lod(dist) (6 - floor(6 * func(lodF, (dist / DFAC))))

VisibilityMode terrainVisibilityMode;
CameraPosition cameraPos;
float fov;
int steps;

module Yardstick(int);
module YardstickI(int);
module Water();
module VNTGerminator(int); // Ventarura
    Germinator (Delay)
module VNTApex(float, float, int, float, bool); // Ventarura Apex
    (Age, Length, NumberOfNodes, Vigour, Fertile)
module VNTInternode(float, float, float, float, float); // Ventarura
    Internode (BaseAge, Age, Length, Vigour, Width)
module VNTSporangium(); // Ventarura
    Sporangium
module VNTHalfSporangium(float, float, float); // Ventarura Half
    Sporangium (Segment, Age, Length, BendFactor)
module Stop();
module VNTBend(float, float);  // Bend (Age, Angle) // doesn’t need to be planet peolkmic?

module VNTBase(float, V3f);  // Ventarura Base (Age, Position)

bool InsideView(V3f point) {
  //return true; // Speed comparison with InsideView off

  V3f camDirReverse = point - cameraPos.position;
  camDirReverse.Normalize();

  //printf("-----------------------------------------------\n");
  //printf("Point at (%f,%f,%f)\n", point.x, point.y, point.z);
  //printf("Direction to camera is (%f,%f,%f)\n", camDirReverse.x, camDirReverse.y, camDirReverse.z);
  //printf("Heading of camera is (%f,%f,%f)\n", cameraPos.head.x, cameraPos.head.y, cameraPos.head.z);
  //printf("Resultant angle is %f vs. a fov of %f\n", cameraPos.head * camDirReverse, abs(cos(fov/2.0)));

  return cameraPos.head * camDirReverse > abs(cos(fov/1.8));
}

float DistanceToCamera(V3f point) {
  return sqrt(pow(point.x - cameraPos.position.x, 2.0f) + pow(point.y - cameraPos.position.y, 2.0f) +
}

Start: {
  cameraPos = GetCameraPosition(0);
  terrainVisibilityMode = Shaded;
  terrainVisibilityAll(terrainVisibilityMode);

  // More code...
}
steps = 0;
}

StartEach: {
cameraPos = GetCameraPosition(0);
fov = 2.0 * atan(1.0/cameraPos.scale * tan(M_PI/4.0));
steps++;
}

derivation length: 400;
Axiom: SetColor(12) Terrain(cameraPos) Water()
/*SB() Yardstick(0) EB()*/ CurrentTexture(2)
SB() VNTGerminator(3) EB();

/******************************************/
production:

Water(): {
produce SetColor(5)
SP() MoveTo(-WTSZ,WTHT,-WTSZ) PP() MoveTo(WTSZ,WTHT,-WTSZ) PP()
MoveTo(WTSZ,WTHT,WTSZ) PP() MoveTo(-WTSZ,WTHT,WTSZ) PP() EP();
}

// A ruler for measuring plant height (and width)
Yardstick(c): {
if (c == 0) {
V3f pointOnTerrain;
V3f initPoint = V3f(-5,0,-5);

bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
if (hitTerrain) {
Printf("Yardstick Point: %f %f
 sill

184
produce MoveTo3f(pointOnTerrain) Yardstick(1);
} else {
  Printf("Point of FALIURE: %f %f %f
",initPoint.x,initPoint.y,initPoint.z);
  Printf("Someone dropped the yardstick off a cliff\n");
  produce ;
}
} else if (c <= 20) {
  produce YardstickI(c) Yardstick(c+1);
}

// Drops a spore randomly on the terrain
VNTGerminator(delay): {
  if (delay > 0) produce VNTGerminator(delay-1);

  V3f pointOnTerrain;
  V3f initPoint = V3f(ranb(-RNGT,RNGT),0,ranb(-RNGT,RNGT));

  initPoint = V3f(0,0,0); // TEST

  bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
  pointOnTerrain.y -= 0.5;
  if (pointOnTerrain.y < WTHT) {
    Printf("Ventarura spore drowned at: %f %f %f
         %f\n",initPoint.x,initPoint.y,initPoint.z);
    return ;
  }
  if (hitTerrain) {
    float vigour = VNT_IVIG;
    Printf("Ventarura germinated at: %f %f %f Vigour: %f,
         %f
",pointOnTerrain.x,pointOnTerrain.y,pointOnTerrain.z);
consider: GetHead VNTBase;

VNTBase(baseAge, pos) < VNTApex(age, length, n, vigour, fertile) >

GetHead(xh,yh,zh): {
if (vigour < 0.25) return;

if (n >= VNT_BRAG*vigour && vigour > 0.45 && ran(1) < (n*vigour)/250.0) {
    // TEMP TEST TOSH
    if (fertile && vigour < 0.3) nproduce SB() RollR(nran(90,1)*n) Right(VNT_SANG)
    VNTSporangium() EB();
}
produce VNTInternode(baseAge, age,length,vigour,VNT_IWD) GetPos(0,0,0)
SB() Left(VNT_BANG) VNTApex(0,0,0,vigour,fertile&&(int)ran(8*vigour))
GetHead(0,0,0) EB() SB() Right(VNT_BANG)
VNTApex(0,0,0,vigour,fertile&&(int)ran(8*vigour)) GetHead(0,0,0) EB();
} else if (age >= VNT_NNAG) {
if (fertile && vigour < 0.3) {
nproduce SB() RollR(ran(90,1)*n) Right(VNT_SANG) VNTSporangium() EB();
produce VNTInternode(baseAge, age,length,vigour,VNT_IWD) GetPos(0,0,0)
    VNTBend(baseAge, VNT_DANG) VNTApex(0,0,n+1,vigour,fertile);
    } else {  
printf("Point of FALIURE: %f %f
%f\n",initPoint.x,initPoint.y,initPoint.z);
printf("The spore was falling to the ground, but then it saw something so
surprising that it missed\n");
produce SetColor(23) MoveTo3f(initPoint) Sphere(1) F(3) Sphere(1) F(3)
Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1)
F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1); //Test
}  


} else {
produce VNTInternode(baseAge, age, length, vigour, VNT_IWWD) GetPos(0,0,0)
    RollR(VNT_RROT) VNTBend(baseAge, VNT_DANG)
    VNTApex(0,0,n+1, vigour, fertile);
}

produce VNTApex(age+dt, length+growthRate(age)*min(vigour,1.0f), n,
    vigour*vigourDecay(yh), fertile);
}

// Pipe model
consider: VNTInternode;

VNTInternode(baseAge, age, length, vigour, width) > SB() VNTInternode
    (i0, i1, i2, i3, widthRL) EB() SB() VNTInternode (i4, i5, i6, i7,
    widthRR) EB(): {
    float deltaT = dt;
    produce VNTInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f), vigour,
        sqrt(pow(widthRL,2.0f)+pow(widthRR,2.0f)));
}

VNTInternode(baseAge, age, length, vigour, width) > VNTInternode (i0, i1,
    i2, i3, widthR): {
    float deltaT = dt;
    produce VNTInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f), vigour, widthR*1.015);
}

VNTInternode(baseAge, age, length, vigour, width): {
    float deltaT = dt;
    produce VNTInternode(baseAge+deltaT, age+deltaT,
        length+growthRate(age)*min(vigour,1.0f), vigour, width);
VNTSporangium(): {
    produce SetColor(6) Elasticity(-0.05) EndGC()
    CurrentContour(halfSporangium)
    SB() StartGC() VNTHalfSporangium(0, nran(1,0.02), nran(1,0.25))
    GetPos(0,0,0) EndGC() EB()
    SB() RollR(180) StartGC() VNTHalfSporangium(0, nran(1,0.02),
    nran(1,0.25)) GetPos(0,0,0) EndGC() EB();
}

VNTHalfSporangium(age, length, bendFactor): {
    produce VNTHalfSporangium(age+dt, length, bendFactor);
}

consider: Stop;

VNTBend(age, ang): {
    if (InRightContext(Stop())) produce VNTBend(age+dt, ang);
    else if (InRightContext(SB() Stop() EB()) || InRightContext(SB() EB())
        SB() Stop() EB())
    produce VNTBend(age+dt, ang) Stop();

    if (age > DEATH) produce VNTBend(age+dt, ang*(1+(0.0175*shrivel(age))));
    else produce VNTBend(age+dt, ang);
}

VNTBase(age, pos): {
    produce VNTBase(age+dt, pos);
}

/*****************************/
interpretation:
maximum depth: 24;

consider: GetPos;

YardstickI(c): {
if (c % 10 == 0) produce SetWidth(1) SetColor(15) F(1);
else produce SetWidth(1) SetColor(22 + c % 2) F(1);
}

VNTApex(age, length, n, vigour, fertile): {
produce SetWidth(0.04) F(0.1) EndGC() Sphere(0.02);
}

VNTInternode(baseAge, age, length, vigour, width) > GetPos(x, y, z): {
if (InsideView(V3f(x, y, z)))
produce SetColor(colour(age)) SetWidth(width) F(length) PointGC();
else produce EndGC() f(length) StartGC();
}

VNTHalfSporangium(age, length, bendFactor) > GetPos(x, y, z): {
if (InsideView(V3f(x, y, z))) {
int numSegments = lod(DistanceToCamera(V3f(x, y, z))) * 4;
float sFac = 24 / (float) numSegments;
float sFl = numSegments;
float sHa = numSegments / 2.0;
for (int n = 0; n < numSegments; n++)
produce
    SetWidth(max(min(age, 8.0f) / 8.0f, 0.4f) * VNT_IWDD * sporangiumWidth(n / sFl))
    Right(bendFactor * sporangiumSplit(n * sFac, age))
F(max(min(age, 8.0f) / 8.0f, 0.2f) * length * VNT_SPLN * ((sFl + 1 - n) / sHa) / sFl);
produce ;
} else produce ;
A.6 Trichopherophyton

#include <algorithm>
#include <cstdio>
#include <string>
#include <vector>
#include <iostream>
#include "lpfgall.h"
using namespace std;

#define nran(mean, sd)
    (((sqrt(-2 * log(ran(1))) * cos(6.283185307 * ran(1))) * sd + mean) // Normally
distributed random number with given mean and sd
#define ranb(low, high) (ran((high) - (low)) + (low))
#define sign(x) ((x > 0.0) - (x < 0.0))

#define WTSZ (80) // Water size
#define WTHT (1.5) // Water height
#define RNGT (75) // Range on terrain where
germinator can appear; max 75 to stay on terrain at scale 5
# define DEATH 40  // Base age at which the plant dies
#define DFAC (25*cameraPos.scale)  // Distance factor for lod
#define DELT 0.08  // Non-Varying Timestep

#define TRC_NNAG (0.18)  // Age to generate a new node
together with other conditions // not used?
#define TRC_SPAG (0.16)  // Age to generate sporangium

#define TRC_BRAG (12)  // Age to branch
#define TRC_BANG nran(15,3)  // Branching Angle
#define TRC_DANG nran(0,3)  // Deflection Angle
#define TRC_RROT nran(137.5/TRC_BRAG,5)  // Roll Branches
#define TRC_SANG nran(60,9)  // Sporangium angle

#define TRC_IVIG nran(1.0,0.1)  // Initial vigour value
#define TRC_SPLN (0.2)  // TRCSporangium Length
#define TRC_IIWD (0.075)  // Initial internode width
#define TRC_IVAN (30)  // Initial vernation angle // not used?
#define TRC_NSFR (8)  // Number of spinose projections per internode per detail level

#define dt nran(DELT,DELT/2.0)  // Varying Timestep
#define dtt nran(DELT,DELT/20.0)  // Varying Timestep (Less Nutso)

    // Jeepers, this code is hacky.

#define growthRate(age) (func(growthRateF,age)*dt*nran(0.65,0.15))
define growthRateFixed(age) (func(growthRateF,age)*DELT*0.65)
define colour(a) floor(36+71*func(colourF,(a/60.0)))
define vigourDecay(yh)
pow(func(vigourDecayF,ran(1))*(pow(yh,0.125f)),0.125f)
define shrivel(age) func(shrivelF,(age-DEATH)/20.0)
define sporangiumWidth(n) (func(sporangiumWidthF,n)*5.0)
define sporangiumSplit(n,age)
((pow(abs(func(sporangiumSplitF,age/20.0f)*((12.0f-(n*min(age,75.0f)/20.0f))/0.75f)),0.4f)*sign((12.0f-(n*min(age,75.0f)/20.0f))))

#define vernationAngle(age) (func(vernationF,age)*100)
#define unfurl(age) (func(unfurlF,age/15.0))
#define lod(dist) (6-floor(6*func(lodF,(dist/DFAC))))

VisibilityMode terrainVisibilityMode;
CameraPosition cameraPos;
float fov;
int steps;

module Yardstick(int);
module YardstickI(int);
module TRCGerminator(int);  // Trichopherophyton Germinator (Delay)
module TRCApex(float, float, int, float, bool);  // Trichopherophyton Apex (Age, Length, NumberOfNodes, Vigour, Fertile)
module TRC Vernation(float, float, float, int, int, bool);  // Trichopherophyton Vernation (Age, Length, Vigour, NumberOfNodes, Original NumberOfNodes, Fertile)
module TRC Internode(float, float, float, float, float, bool);  // Trichopherophyton Internode (BaseAge, Age, Length, Vigour, Width, Fixed Growth)
module TRCSporangium(int);  // Trichopherophyton Sporangium (Delay)
module TRCHalfSporangium(float, float, float);  // Trichopherophyton Half Sporangium (Age, Length, BendFactor)
module Stop();
module TRCBend(float, float);  // Bend (Age, Angle) // Doesn't need to be plant specific?
module TRCVernationBend(float, float, int, int, int);  // Trichopherophyton Vernation Bend (Age, Angle, NodeNumber, Target NodeNumber, Total NodeNumber)
module TRCBase(float, V3f); //

    Trichopherophyton Base (Age, Position)

bool InsideView(V3f point) {
    //return true; // Speed comparison with InsideView off
    
    V3f camDirReverse = point - cameraPos.position;
    camDirReverse.Normalize();

    //printf("------------------------------------------\n");
    //printf("Point at (%f,%f,%f)\n", point.x, point.y, point.z);
    //printf("Direction to camera is (%f,%f,%f)\n", camDirReverse.x,
    //    camDirReverse.y, camDirReverse.z);
    //printf("Heading of camera is (%f,%f,%f)\n", cameraPos.head.x,
    //    cameraPos.head.y, cameraPos.head.z);
    //printf("Resultant angle is %f vs. a fov of %f\n",
    //    cameraPos.head*camDirReverse, abs(cos(fov/2.0)));

    return cameraPos.head*camDirReverse > abs(cos(fov/1.8));
}

float DistanceToCamera(V3f point) {
    return
        sqrt(pow(point.x-cameraPos.position.x,2.0f)+pow(point.y-cameraPos.position.y,2.0f)+

Start: {
    terrainVisibilityMode = Shaded;
    terrainVisibilityAll(terrainVisibilityMode);
    steps = 0;
}

193
StartEach: {
cameraPos = GetCameraPosition(0);
fov = 2.0 * atan(1.0/cameraPos.scale * tan(M_PI/4.0));
steps++;
}

derivation length: 400;
Axiom: SetColor(12) Terrain(cameraPos)
/*SB() Yardstick(0) EB()*/ CurrentTexture(2)
SB() TRCGerminator(3) EB();

/**********************************/
production:

Terrain(t): {
cameraPos = GetCameraPosition(0);
produce Terrain(cameraPos) SetColor(5)
SP() MoveTo(-WTSZ,WTHT,-WTSZ) PP() MoveTo(WTSZ,WTHT,-WTSZ) PP()
MoveTo(WTSZ,WTHT,WTSZ) PP() MoveTo(-WTSZ,WTHT,WTSZ) PP() EP();
}

// A ruler for measuring plant height (and width)
Yardstick(c): {
if (c == 0) {
V3f pointOnTerrain;
V3f initPoint = V3f(-5,0,-5);

bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
if (hitTerrain) {
Printf("Yardstick Point: %f %f
%f
",pointOnTerrain.x,pointOnTerrain.y,pointOnTerrain.z);
produce MoveTo3f(pointOnTerrain) Yardstick(1);
} else {
    printf("Point of FAILURE: %f %f %f\n", initPoint.x, initPoint.y, initPoint.z);
    printf("Someone dropped the yardstick off a cliff\n");
    produce ;
}
} else if (c <= 20) {
    produce YardstickI(c) Yardstick(c+1);
}
}

// Drops a spore randomly on the terrain
TRCGerminator(delay): {
    if (delay > 0) produce TRCGerminator(delay-1);

    V3f pointOnTerrain;
    V3f initPoint = V3f(ranb(-RNGT,RNGT),0,ranb(-RNGT,RNGT));

    initPoint = V3f(0,0,0); // TEST

    bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
    pointOnTerrain.y -= 0.5;
    if (pointOnTerrain.y < WTHT) {
        printf("Trichopherophyton spore drowned at: %f %f %f\n", initPoint.x, initPoint.y, initPoint.z);
        return ;
    }
    if (hitTerrain) {
        float vigour = TRC_IVIG;
        printf("Trichopherophyton germinated at: %f %f %f Vigour: %f (%i)\n", pointOnTerrain.x, pointOnTerrain.y, pointOnTerrain.z, vigour, steps);
        produce RollR(ran(360)) TRCBase(0, pointOnTerrain) StartGC()
TRCApex(0,0,0,vigour,true) GetHead(0,0,0);

} else {

    Printf("Point of FALIURE: %f %f %f\n", initPoint.x, initPoint.y, initPoint.z);
    Printf("The spore was falling to the ground, but then it saw something so surprising that it missed\n");
    produce SetColor(23) MoveTo3f(initPoint) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1); //Test
}

consider: GetHead TRCBASE;

TRCBASE(baseAge, pos) < TRCApex(age, length, n, vigour, fertile) >
    GetHead(xh,yh,zh): {
if (vigour < 0.4) {
    int vernationSize = nran(20,4);
    produce TRCVernation(0,0,vigour,vernationSize,vernationSize,fertile);
}
if (n > TRC_BRAG && vigour > 0.45 && ran(1) < 0.075) { // Dichotomous Branch
produce TRCInternode(baseAge,age,length,vigour,TRC_IIWD,false)
    GetPos(0,0,0) SB() Left(TRC_BANG)
    TRCApex(0,0,0,vigour,fertile&&((int)ran(4))) GetHead(0,0,0) EB() SB()
    Right(TRC_BANG) TRCApex(0,0,0,vigour,fertile&&((int)ran(4)))
    GetHead(0,0,0) EB();
} else if (n > TRC_BRAG && vigour > 0.45 && ran(1) < 0.225) { // Pseudomonopodial Branch
produce TRCInternode(baseAge,age,length,vigour,TRC_IIWD,false)
    GetPos(0,0,0) SB() Right(TRC_BANG*1.5)
    TRCApex(0,0,0,vigour,fertile&&((int)ran(4))) GetHead(0,0,0) EB() SB()
TRCApex(0,0,0,vigour,fertile&& (int) ran(4)) GetHead(0,0,0) EB();
} else if (age >= TRC_NNAG) {
    produce TRCInternode(baseAge,age,length,vigour,TRC_IIWD,false)
    GetPos(0,0,0) RollR(TRC_RROT) TRCBend(baseAge, TRC_DANG)
    TRCApex(0,0,n+1,vigour,fertile);
}
produce TRCApex(age+dt, length+growthRate(age)*min(vigour,1.0f), n,
    vigour*vigourDecay(yh), fertile);
}

TRCBase(baseAge, pos) < TRCVernation(age,length,vigour,n,nO,fertile): {
    if (age >= TRC_NNAG && n > 0) {
        if (vigour < 0.3 && fertile) nproduce SB() RollR(90*n) Right(TRC_SANG)
            TRCSporangium(n*(TRC_NNAG/DELT)*2) EB();
        produce TRCVernationBend(0, vernationAngle(age),0,n,nO)
            TRCInternode(baseAge,age,0.0,vigour,TRC_IIWD,true) GetPos(0,0,0)
        TRCVernation(0,length,vigour,n-1,nO,fertile);
    }
    produce TRCVernation(age+dtt, length, vigour*vigourDecay(1),n,nO,fertile);
}

// Pipe model
consider: TRCInternode;

TRCInternode(baseAge, age, length, vigour, width, fg) > SB() TRCInternode
    (i0, i1, i2, i3, widthRL, i4) EB() SB() TRCInternode (i5, i6, i7, i8,
    widthRR, i9) EB(): {
    float deltaT = dt;
    if (fg) produce TRCInternode(baseAge+DELT, age+DELT,
        length+growthRateFixed(age)*min(vigour,1.0f), vigour,
        sqrt(pow(widthRL,2.0f)+pow(widthRR,2.0f)),fg);
    else produce TRCInternode(baseAge+deltaT, age+deltaT,
        length+growthRate
length + growthRate(age) * min(vigour, 1.0f), vigour,
    sqrt(pow(widthRL, 2.0f) + pow(widthRR, 2.0f)), fg);
}
TRCInternode(baseAge, age, length, vigour, width, fg) > TRCInternode (i0,
    i1, i2, i3, widthR, i4): {
    float deltaT = dt;
    if (fg) produce TRCInternode(baseAge + DELT, age + DELT,
        length + growthRateFixed(age) * min(vigour, 1.0f), vigour, widthR * 1.015,
        fg);
    else produce TRCInternode(baseAge + deltaT, age + deltaT,
        length + growthRate(age) * min(vigour, 1.0f), vigour, widthR * 1.015, fg);
}
TRCInternode(baseAge, age, length, vigour, width, fg): {
    float deltaT = dt;
    if (fg) produce TRCInternode(baseAge + DELT, age + DELT,
        length + growthRateFixed(age) * min(vigour, 1.0f), vigour, width, fg);
    else produce TRCInternode(baseAge + deltaT, age + deltaT,
        length + growthRate(age) * min(vigour, 1.0f), vigour, width, fg);
}
TRCSporangium(delay): {
    if (delay > 0) produce TRCSporangium(delay - 1);
    produce SetColor(6) Elasticity(-0.1) EndGC()
    CurrentContour(halfSporangium)
    SB() StartGC() TRCHalfSporangium(0, nran(1, 0.02), nran(1, 0.25))
    GetPos(0, 0, 0) EndGC() EB()
    SB() RollR(180) StartGC() TRCHalfSporangium(0, nran(1, 0.02),
        nran(1, 0.25)) GetPos(0, 0, 0) EndGC() EB();
}
TRCHalfSporangium(age, length, bendFactor): {
    produce TRCHalfSporangium(age + dt, length, bendFactor);
consider: Stop;

TRCBend(age, ang): {
if (InRightContext(Stop())) produce TRCBend(age+dt, ang);
else if (InRightContext(SB() Stop() EB()) || InRightContext(SB() EB()  
  SB() Stop() EB()))
produce TRCBend(age+dt, ang) Stop();

if (age > DEATH) produce TRCBend(age+dt, ang*(1+(0.0175*shrivel(age))));
else produce TRCBend(age+dt, ang);
}

TRCVernationBend(age, ang, n, nT, nO): {
if (age >= TRC_NNAG*0.75 && n < nT)
produce TRCVernationBend(0, vernationAngle((nO-n)/(nO*1.5)),n+1,nT,nO);
else if (n == nT) produce TRCVernationBend(DELT*n,ang,n+1,nT,nO); //  
    HUGE HACK
produce TRCVernationBend(age+dtt*1.5,ang,n,nT,nO);
}

TRCBase(age, pos): {
produce TRCBase(age+dt, pos);
}

/*****************************/
interpretation:
maximum depth: 24;

consider: GetPos;
YardstickI(c): {
if (c % 10 == 0) produce SetWidth(1) SetColor(15) F(1);
else produce SetWidth(1) SetColor(22 + c % 2) F(1);
}

TRCApex(age, length, n, vigour, fertile): {
produce SetWidth(0.04) F(0.1) EndGC() Sphere(0.02);
}

TRCVernation(age, length, vigour, n, n0, fertile): {
produce SetWidth(0.04) F(0.02) EndGC() Sphere(0.02);
}

TRCIternode(baseAge, age, length, vigour, width, fg) > GetPos(x,y,z): {
if (InsideView(V3f(x,y,z))) {
int numProjections = TRC_NSPR * lod(DistanceToCamera(V3f(x,y,z))) - 16;
for (int i = 0; i < numProjections; i++)
produce SB() f(length * (i / (float) numProjections)) RollR(137.5 * i)
Right(90) SetWidth(0.015) PointGC() SetWidth(0.001)
F((width / 4.0 * ranb(0.2, 1)) + (width / 2.0)) EndGC() EB();
produce SetColor(colour(age)) SetWidth(width) F(length) PointGC();
} else produce EndGC() f(length) StartGC();
}

TRCHalfSporangium(age, length, bendFactor) > GetPos(x,y,z): {
if (InsideView(V3f(x,y,z))) {
int numSegments = lod(DistanceToCamera(V3f(x,y,z))) * 4;
float sFac = 24 / (float) numSegments;
float sFl = numSegments;
float sHa = numSegments / 2.0;
for (int n = 0; n < numSegments; n++)
produce
}
```c
SetWidth(max(min(age/2.0f,8.0f)/8.0f,0.4f)*TRC_IWD*2.0*sporangiumWidth(n/sFl))
Right(bendFactor*sporangiumSplit(n*sFac,age))
F(max(min(age/2.0f,8.0f)/8.0f,0.2f)*length*TRC_SPLN*((sFl+1-n)/sHa)/sFl);
produce ;
} else produce ;
}

TRCBend(age, ang): {
produce Up(ang);
}

TRCVernationBend(age,ang,n,nT,nO): {
produce Down(ang*unfurl(age));
}

TRCBase(age, pos): {
//produce MoveTo3f(pos); // Speed comparison with lod off
produce MoveTo3f(pos) SetWidth(0.01)
      ContourSides(lod(DistanceToCamera(pos))*3);
}

A.7  Asteroxylon

#include <algorithm>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include "lpfgall.h"
using namespace std;

#define nran(mean, sd)
```
\[
\left(\sqrt{-2\log(\text{ran}(1))}\cos(6.283185307\cdot\text{ran}(1))\right)\cdot\text{sd+mean} \quad \text{// Normally distributed random number with given mean and sd}
\]

\#define ranb(low,high) (\text{ran}((\text{high})-(\text{low}))+\text{(low)})

\#define WTSZ (80) \quad \text{// Water size}
\#define WTHT (1.5) \quad \text{// Water height}
\#define RNGT (75) \quad \text{// Range on terrain where germinator can appear; max 75 to stay on terrain at scale 5}
\#define DEATH (40) \quad \text{// Base age at which the plant dies}
\#define DFAC (25\cdot\text{cameraPos.scale}) \quad \text{// Distance factor for lod}
\#define DELT 0.08 \quad \text{// Non-Varying Timestep}

\#define AST_NNAG (0.24) \quad \text{// Age to generate a new node}
\#define AST_BRAG (18) \quad \text{// Age to branch}
\#define AST_BANG nran(50,5) \quad \text{// Branching Angle}
\#define AST_DANG nran(0,2) \quad \text{// Deflection Angle}
\#define AST_RROT nran(137.5/AST_BRAG,5) \quad \text{// Roll Branches // Not used???
\#define AST_SANG nran(60,6) \quad \text{// ASTSporangium angle}
\#define AST_IVIG nran(1.3,0.3) \quad \text{// Initial vigour value}
\#define AST_RANG (137.5) \quad \text{// Phyllotactic angle}
\#define AST_ENAN nran(85,3) \quad \text{// ASTEnation angle}
\#define AST_MAXO (4.0) \quad \text{// Predicted maximum order}
\#define AST_RANE (\text{ran}(2.999999)) \quad \text{// Random enation surface}

\#define dt nran(DELT,DELT/2.0) \quad \text{// Varying Timestep}

\#define growthRate(age) (func(growthRateF,\text{age})\cdot\text{dt}\cdot\text{nran}(0.65,0.15))
\#define vigourDecay(order)
\quad (1.0-((1.0-\text{pow}(\text{func(vigourDecayF,ran(1)),0.05f})\cdot\text{dt}\cdot15.0\cdot(\text{order+1})))
\#define sporangiumWidth(n) (func(sporangiumWidthF,n)*5.0)
\#define shrivel(age) func(shrivelF,(\text{age}-\text{DEATH})/20.0)

202
#define width(n,o) (func(widthF,1.15-n)*func(orderReductionF,o/AST_MAXO))
#define scale(n,o)
    ((0.5+func(widthF,1-n)/2.0)*func(orderReductionF,(o-1)/AST_MAXO)*0.5)
#define enationBend(n) (func(enationBendF,1.15-n))
#define elasticity(o) ((0.12-o*0.08)*DELT)
#define age(a) (func(ageWidthF,a/20.0))
#define ageBend(a) (func(ageBendF,a/20.0)/2.0)
#define bend(n,a) (func(clampF,enationBend(n)+ageBend(a)))
#define enationGrowth(a) (func(enationGrowthF,a/20.0))
#define colour(n,o,a) (107-71*func(clampF,(((1-(a/30.0))*(1.55-n))+1.5)/2)
#define lod(dist) (6-floor(6*func(lodF,(dist/DFAC))))

VisibilityMode terrainVisibilityMode;
CameraPosition cameraPos;
float fov;
int steps;

module Yardstick(int);
module YardstickI(int);
module Water();
module ASTGerminator(int);     // Asteroxylon Germinator (Delay)
module ASTBase(float, V3f);     // Asteroxylon Base (Age, Position)
module ASTApex(float, float, int, float, int);     // Asteroxylon Apex (Age, Length, NumberOfNodes, Vigour, Order)
module ASTInternode(float, float, float, float, int);     // Asteroxylon Internode (Age, Length, Vigour, Width, Order)
module ASTEnation(float, float, float, int, float, int);     // Asteroxylon Enation (Age, Scale, Vigour, Order, Angle, Random Surface)
module ASTSporangium(float, float, int);
bool InsideView(V3f point) {
    // return true; // Speed comparison with InsideView off
    V3f camDirReverse = point - cameraPos.position;
    camDirReverse.Normalize();
    return cameraPos.head * camDirReverse > abs(cos(fov / 1.8));
}

float DistanceToCamera(V3f point) {
    return sqrt(pow(point.x - cameraPos.position.x, 2.0f) + pow(point.y - cameraPos.position.y, 2.0f) +
    pow(point.z - cameraPos.position.z, 2.0f));
}

Start: {
    cameraPos = GetCameraPosition(0);
    terrainVisibilityMode = Shaded;
    terrainVisibilityAll(terrainVisibilityMode);
    steps = 0;
}

StartEach: {
    cameraPos = GetCameraPosition(0);
    fov = 2.0 * atan(1.0 / cameraPos.scale * tan(M_PI / 4.0));
    steps ++;
}

derivation length: 770;
Axiom: SetColor(12) Terrain(cameraPos) Water()
/* SB() Yardstick(0) EB() */
SB() ASTGerminator(3) EB();
production:

Water(): {
produce SetColor(5)
SP() MoveTo(-WTSZ,WTHT,-WTSZ) PP() MoveTo(WTSZ,WTHT,-WTSZ) PP()
MoveTo(WTSZ,WTHT,WTSZ) PP() MoveTo(-WTSZ,WTHT,WTSZ) PP() EP();
}

// A ruler for measuring plant height (and width)
Yardstick(c): {
if (c == 0) {
V3f pointOnTerrain;
V3f initPoint = V3f(-5,0,-5);
bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
if (hitTerrain) {
Printf("Yardstick Point: %f %f
%f\n", pointOnTerrain.x, pointOnTerrain.y, pointOnTerrain.z);
produce MoveTo3f(pointOnTerrain) Yardstick(1);
} else {
Printf("Point of FALIURE: %f %f
%f\n", initPoint.x, initPoint.y, initPoint.z);
Printf("Someone dropped the yardstick off a cliff\n");
produce ;
}
} else if (c <= 40) {
produce YardstickI(c) Yardstick(c+1);
}
}

// Drops a spore randomly on the terrain
ASTGerminator(delay): {
    if (delay > 0) produce ASTGerminator(delay-1);
}

V3f pointOnTerrain;
V3f initPoint = V3f(ranb(-RNGT,RNGT),0,ranb(-RNGT,RNGT));

//initPoint = V3f(0,0,0); // TEST

bool hitTerrain = terrainHeightAt(initPoint, pointOnTerrain);
pointOnTerrain.y -= 0.5;
if (pointOnTerrain.y < WTHT) {
    Printf("Asteroxylon spore drowned at: %f %f %f \n",initPoint.x,initPoint.y,initPoint.z);
    return;
}
if (hitTerrain) {
    float vigour = AST_IVIG;
    Printf("Asteroxylon germinated at: %f %f %f Vigour: %f (%i) \n",pointOnTerrain.x,pointOnTerrain.y,pointOnTerrain.z,vigour,steps);
    produce ASTBase(0, pointOnTerrain) StartGC() ASTApex(0, 0, -12, vigour, 0);
} else {
    Printf("Point of FALIURE: %f %f %f \n",initPoint.x,initPoint.y,initPoint.z);
    Printf("The spore was falling to the ground, but then it saw something so surprising that it missed\n");
    produce SetColor(23) MoveTo3f(initPoint) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1) F(3) Sphere(1); //Test
}
}
ASTApex(age, length, n, vigour, order): {
if (vigour <= 0.15) return;
if (n >= AST_BRAg/(order/6.0+1.0) && vigour > 0.35 && ran(1) < 0.05) {
    produce ASTInternode(age, length, vigour, 0.05, order) GetPos(0,0,0) SB()
    Left(AST_BANG) Elasticity(elasticity(order))
    ASTApex(0,0,0, vigour+0.15, order+1) EB()
}
ASTApex(age+dt, length+growthRate(age)*min(vigour,1.0f), n, vigour*vigourDecay(order), order);
} else if (age >= AST_NNAG) {
    produce ASTInternode(age, length, vigour, 0.05, order) GetPos(0,0,0)
    RollR(AST_RANG/6.0) Up(AST_DANG) ASTApex(0,0,n+1, vigour, order);
} produce ASTApex(age+dt, length+growthRate(age)*min(vigour,1.0f), n, vigour*vigourDecay(order), order);
}

consider: ASTEnation;
ASTInternode(age, length, vigour, width, order): {
float i1, i2, i3, i5; int i4, i6;
if (InRightContext(SB()) ASTEnation(i1,i2,i3,i4,i5,i6) EB())) {
    // printf("I don't think he'll be very keen; he's already got one!\n");
    produce ASTInternode(age+dt, length+growthRate(age)*min(vigour,1.0f),
        vigour, width, order);
} else {
    float factor = (1-vigour);
    float random = ran(factor);
    if (order == 0 && random > 0.6 && ran(1) < 0.4) {
        produce ASTInternode(age+dt, length+growthRate(age)*min(vigour,1.0f),
            vigour, width, order)
        SB() ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
        GetPos(0,0,0) EB()
    ASTSporangium(age, vigour, order) GetPos(0,0,0);
if (vigour > 0.6) {
  nproduce SB() f(length/2.0)
  SB() ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
    EB() GetPos(0,0,0)
  SB() RollR(60)
    ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
    GetPos(0,0,0) EB()
  SB() RollR(120)
    ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
    GetPos(0,0,0) EB()
  SB() RollR(180)
    ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
    GetPos(0,0,0) EB()
  SB() RollR(240)
    ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
    GetPos(0,0,0) EB()
  SB() RollR(300)
    ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
    GetPos(0,0,0) EB();
}
produce ASTInternode(age+dt, length+growthRate(age)*min(vigour,1.0f),
                        vigour, width, order)
SB()
  ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
    GetPos(0,0,0) EB()
SB() RollR(60)
  ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
    GetPos(0,0,0) EB()
SB() RollR(120)
  ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
    GetPos(0,0,0) EB()
SB() RollR(180)
ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
GetPos(0,0,0) EB()

SB() RollR(240)
ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
GetPos(0,0,0) EB()

SB() RollR(300)
ASTEnation(age, scale(vigour, order), vigour, order, AST_ENAN, AST_RANE)
GetPos(0,0,0) EB();

ASTEnation(age, scale, vigour, order, angle, random): {
produce ASTEnation(age+dt, scale, vigour, order, angle, random);
}

ASTBase(age, pos): {
produce ASTBase(age+dt, pos);
}

interpretation:
maximum depth: 25;

consider: GetPos;

YardstickI(c): {
if (c%10 == 0) produce SetWidth(1) SetColor(15) F(1);
else produce SetWidth(1) SetColor(22+c%2) F(1);
}

ASTApex(age, length, n, vigour, order): {
produce EndGC();
}

ASTInternode(age, length, vigour, width, order) > GetPos(x,y,z): {
  if (InsideView(V3f(x,y,z)))
  produce SetColor(colour(vigour,order,age))
      SetWidth(width(vigour,order)*age(age)) F(length) PointGC();
  else produce EndGC() f(length) StartGC();
}

ASTEnation(age, scale, vigour, order, angle, random) > GetPos(x,y,z): {
  if (InsideView(V3f(x,y,z)))
  produce SetColor(colour(vigour,order,age)) Right(90)
      f(width(vigour,order)*age(age)*0.5)
  Left(angle+bend(vigour,age)*20) RollL(90)
  Surface(2+random, scale*0.6*enationGrowth(age));
  else produce ;
}

ASTSporangium(age, vigour, order) > GetPos(x,y,z): {
  if (InsideView(V3f(x,y,z))) {
    float angle = AST_SANG;
    produce SB() RollR(180) f(-0.1)
    ASTEnation(age, scale(vigour,order),vigour,order,angle+7,AST_RANE)
    GetPos(0,0,0) EB()
    SB() RollR(180) Right(90) F(width(vigour,order)*age(age)) Left(angle)
    RollL(90) SetWidth(0.05) PointGC() F(0.65*enationGrowth(age))
    PointGC() SetColor(6) Surface(0, 0.65*enationGrowth(age)) EB();
  } else produce ;
}

ASTBase(age, pos): {
//produce MoveTo3f(pos); // Speed comparison with lod off
produce MoveTo3f(pos) SetWidth(0.01)
    ContourSides(lod(DistanceToCamera(pos))*3);
}

A.8 Ecosystem Simulation

#include <algorithm>
#include <cstdio>
#include <string>
#include <vector>
#include <iostream>
#include "lpfgall.h"
using namespace std;

#define nran(mean,sd)
    ((sqrt(-2*log(ran(1)))*cos(6.283185307*ran(1)))*sd+mean) // Normally
distributed random number with given mean and sd
#define ranb(low,high) (ran((high)-(low))+(low))
#define sign(x) ((x > 0.0) - (x < 0.0))

#define DEATH (13*SVIG)
//#define DECAY (DEATH+6)
#define DECAY (DEATH+1)
#define DT (1)
#define SVIG sqrt(vigour)
#define AGLANGLE 42
#define SPORE_RANGE 1.0
#define MIN_VIG 0.2
#define RAN_VIG 1.0
#define AGL_NMSP 2
# define AST_NMSP 1
# define HRN_NMSP 2
# define NTH_NMSP 2
# define RHY_NMSP 2
# define TRC_NMSP 2
# define VNT_NMSP 2

# define AGL_INIT 9
# define AST_INIT 7
# define HRN_INIT 10
# define NTH_INIT 6
# define RHY_INIT 10
# define TRC_INIT 5
# define VNT_INIT 6

#define SIZE 4.0

enum SoilParameter { Moisture, Litter, Sinter };
enum Species { Aglaophyton, Asteroxylon, Horneophyton, Nothia, Rhynia,
        Trichopherophyton, Ventarura };

module Background();
module Starter();
module Generator();

module AGLSpore(V3f, float, int);
module ASTSpore(V3f, float, int);
module HRNSpore(V3f, float, int);
module NTHSpore(V3f, float, int);
module RHYSpore(V3f, float, int);
module TRCSpore(V3f, float, int);
module VNTSpore(V3f, float, int);
typedef unsigned char byte;

int toImgSpace(float value) {
    return (value/SIZE+0.5) * 128;
}

float minMoisturePreference[7] = {0.1, 0.1, 0.7, 0.3, 0.1, 0.4, 0.9};
float maxMoisturePreference[7] = {0.8, 0.6, 1.0, 0.7, 0.5, 0.8, 1.0};
float minLitterPreference[7] = {0.6, 0.5, 0.4, 0.4, 0.1, 0.7, 0.5};
float maxLitterPreference[7] = {1.0, 1.0, 0.8, 0.8, 0.6, 1.0, 1.0};
float minSinterPreference[7] = {0.0, 0.0, 0.2, 0.0, 0.0, 0.0, 0.0};
float maxSinterPreference[7] = {0.2, 0.3, 0.7, 0.2, 0.8, 0.2, 0.2};
float delay[7] = {1.0, 2.0, 0.8, 1.0, 0.7, 2.5, 1.6};
float rarity[7] = {0.85, 0.65, 0.95, 0.60, 1.00, 0.45, 0.55};

// Because these simple plants are not modeled with sporangia, spores are
// spread by apices, which causes some plants to be way more successful
// than others. This factor should correct the imbalance
float sporeProductionMultiplier[7] = { 0.50, 0.10, 1.00, 1.00, 0.50, 1.00, 1.20 };
float speciesRFactor[7] = { 1.25, 1.00, 0.60, 0.85, 1.25, 0.85, 0.65 };
string str;

class SoilEnvironment {
public:
SoilEnvironment (string name = "") { // Insanely bad bmp loading thing
    that only works on exactly 128x128 with the right headers. I hate
    loading images.
    //printf(name.c_str());

    FILE* file;
    file = fopen(name.c_str(), "rb");

    byte garbagePile[54];
    int size = 49152;
    int dimension = 128;
    byte values[49152];
    if(file != NULL) {
        fread(garbagePile, 54, 1, file); // Throw away header
        fread(values, 1, size, file);

        fclose(file);
    } else {
        printf("Kapow!!\n");
        return;
    }

    for (int x = 0; x < dimension; x++) {
        for (int y = 0; y < dimension; y++) {
            int index = (y*dimension+x)*3;
        }
    }
}
blueValues[x][y] = (int)values[index];
greenValues[x][y] = (int)values[index+1];
redValues[x][y] = (int)values[index+2];
}
}
}

float getValue(float x, float y, int channel) { return getValue(toImgSpace(x), toImgSpace(y), channel); }
float getValue(int x, int y, int channel) {
    if (x < 0 || y < 0 || x >= 128 || y >= 128) return 0;
    if (channel == 0) return redValues[x][y]/255.0;
    else if (channel == 1) return greenValues[x][y]/255.0;
    else if (channel == 2) return blueValues[x][y]/255.0;
    else return -1;
}

private:
float redValues[128][128];
float greenValues[128][128];
float blueValues[128][128];
};
class PlantInfo {
    public:
        V3f position;
        int id;
        float radius;
        bool alive;
};

int iteration;
SoilEnvironment* env;
vector<V3f> aglaophytonSpores;
vector<V3f> asteroxylonSpores;
vector<V3f> horneophytonSpores;
vector<V3f> nothiaSpores;
vector<V3f> rhyniaSpores;
vector<V3f> trichopherophytonSpores;
vector<V3f> ventaruraSpores;
vector<PlantInfo*> plantInfo;
int nextID;
int totalAGL, totalAST, totalHRN, totalNTH, totalRHY, totalTRC, totalVNT;
int currentAGL, currentAST, currentHRN, currentNTH, currentRHY,
    currentTRC, currentVNT;

float getVigour(V3f position, Species species) {
    float localMoisture = env->getValue(position.x, position.z, Moisture);
    float localLitter = env->getValue(position.x, position.z, Litter);
    float localSinter = env->getValue(position.x, position.z, Sinter);
    float moistureSuitability, litterSuitability, sinterSuitability, vigour;

    if (localMoisture == 0 || localLitter == 0) return 0;

    if (localMoisture >= minMoisturePreference[species] && localMoisture <=
        maxMoisturePreference[species]) moistureSuitability = 1;
    else moistureSuitability =
        1.0 - sqrt(min(abs(minMoisturePreference[species]-localMoisture),abs(maxMoisturePreference[species]-localMoisture)));

    if (localLitter >= minLitterPreference[species] && localLitter <=
        maxLitterPreference[species]) litterSuitability = 1;
    else litterSuitability =
        1.0 - sqrt(min(abs(minLitterPreference[species]-localLitter),abs(maxLitterPreference[species]-localLitter)));

    if (localSinter >= minSinterPreference[species] && localSinter <=
        maxSinterPreference[species]) sinterSuitability = 1;
    else sinterSuitability =
        1.0 - sqrt(min(abs(minSinterPreference[species]-localSinter),abs(maxSinterPreference[species]-localSinter)));

    vigour = moistureSuitability * litterSuitability * sinterSuitability;
    return vigour;
}
else sinterSuitability = 
    1.0 - sqrt(min(abs(minSinterPreference[species] - localSinter), abs(maxSinterPreference[species] - localSinter)));
return moistureSuitability * litterSuitability * sinterSuitability;
}

def distanceToNearestPlant(V3f position) {
    float minDistanceSquared = 1000000;
    for (int i = 0; i < plantInfo.size(); i++) {
        float distanceSquared = (plantInfo[i]->position.x - position.x) * 
        (plantInfo[i]->position.x - position.x) + (plantInfo[i]->position.z - 
        position.z) * (plantInfo[i]->position.z - position.z);
        if (distanceSquared < minDistanceSquared) minDistanceSquared = 
            distanceSquared;
    }
    return sqrt(minDistanceSquared);
}

PlantInfo* nearestPlant(V3f position, int id) { // Nearest (living) plant 
    (including radius) that is not 'id' (i.e. not itself)
    float minDistance = 1000;
    int current = -1;
    for (int i = 0; i < plantInfo.size(); i++) {
        if (plantInfo[i]->id == id || !plantInfo[i]->alive) continue;
        float distanceSquared = (plantInfo[i]->position.x - position.x) * 
        (plantInfo[i]->position.x - position.x) + (plantInfo[i]->position.z - 
        position.z) * (plantInfo[i]->position.z - position.z);
        float distance = distanceSquared - plantInfo[i]->radius * 
        plantInfo[i]->radius;
        if (distance < minDistance) {
            minDistance = distance;
            current = i;
        }
    }
if (current == -1) {
    printf("No nearest plant found for plant %i (%f, %f)\n", id, position.x, position.z);
    return NULL;
}
return plantInfo[current];

PlantInfo* getPlantInfo(int id) {
    for (int i = 0; i < plantInfo.size(); i++) if (plantInfo[i]->id == id)
        return plantInfo[i];
    return NULL;
}

void addPlant(V3f position) {
    PlantInfo* newInfo = new PlantInfo;
    newInfo->position = position;
    newInfo->id = nextID;
    newInfo->radius = 0.03;
    newInfo->alive = true;
    nextID++;
    plantInfo.push_back(newInfo);
}

void deletePlant(int id) {
    int index = -1;
    for (int i = 0; i < plantInfo.size(); i++) {
        if (plantInfo[i]->id == id) {
            index = i;
            break;
        }
    }
    if (index == -1) {
        // code
printf("Could not delete plant with ID %i, because it does not exist
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n", id);
return;
}
plantInfo.erase(plantInfo.begin()+index);
}

Start: {
iteration = 0;
env = new SoilEnvironment("soil++.bmp");
aglaophytonSpores = vector<V3f>();
asteroxylonSpores = vector<V3f>();
horneophytonSpores = vector<V3f>();
nothiaSpores = vector<V3f>();
rhyniaSpores = vector<V3f>();
trichopherophytosporas = vector<V3f>();
ventaruraSpores = vector<V3f>();
nextID = 0;
totalAGL = totalAST = totalHRN = totalNTH = totalRHY = totalTRC =
totalVNT = 0;
currentAGL = currentAST = currentHRN = currentNTH = currentRHY =
currentTRC = currentVNT = 0;
plantInfo.clear();
}

StartEach: {
iteration++;
if (iteration % 10 == 0 || true) {
printf("--------------------------------------------------\n");
printf("Iteration: %i\n", iteration);
printf("Total plants since start: %i\n", nextID);
printf("Plants present: %i\n", plantInfo.size());

219
printf("Total plants per species: %i %i %i %i %i %i %i\n", totalAGL, 
    totalAST, totalHRN, totalNTH, totalRHY, totalTRC, totalVNT);
printf("Current plants per species: %i %i %i %i %i %i %i\n\n", 
    currentAGL, currentAST, currentHRN, currentNTH, currentRHY, 
    currentTRC, currentVNT);
}
}
derivation length: 0;
Axiom: SB() Background() EB() Starter();
production:

Starter(): {
    V3f position;
    for (int i = 0; i < AGL_INIT; i++) { // Try to place a specimen of each 
        species in 5 places before giving up
        position = V3f(-SIZE/2 + ran(SIZE), 0, -SIZE/2 + ran(SIZE));
        float vigour = getVigour(position, Aglaophyton);
        if (vigour > 0.25) nproduce SB() AGLSpore(position, vigour, 0) EB();
    }
    for (int i = 0; i < AST_INIT; i++) {
        position = V3f(-SIZE/2 + ran(SIZE), 0, -SIZE/2 + ran(SIZE));
        float vigour = getVigour(position, Asteroxylon);
        if (vigour > 0.25) nproduce SB() ASTSpore(position, vigour, 0) EB();
    }
    for (int i = 0; i < HRN_INIT; i++) {
        position = V3f(-SIZE/2 + ran(SIZE), 0, -SIZE/2 + ran(SIZE));
        float vigour = getVigour(position, Horneophyton);
        if (vigour > 0.25) nproduce SB() HRNSpore(position, vigour, 0) EB();
    }
    for (int i = 0; i < NTH_INIT; i++) {
        position = V3f(-SIZE/2 + ran(SIZE), 0, -SIZE/2 + ran(SIZE));
float vigour = getVigour(position, Nothia);
if (vigour > 0.25) nproduce SB() NTHSpore(position, vigour, 0) EB();
}
for (int i = 0; i < RHY_INIT; i++) {
    position = V3f(-SIZE/2 + ran(SIZE), 0, -SIZE/2 + ran(SIZE));
    float vigour = getVigour(position, Rhynia);
    if (vigour > 0.25) nproduce SB() RHYSpore(position, vigour, 0) EB();
}
for (int i = 0; i < TRC_INIT; i++) {
    position = V3f(-SIZE/2 + ran(SIZE), 0, -SIZE/2 + ran(SIZE));
    float vigour = getVigour(position, Trichopherophyton);
    if (vigour > 0.25) nproduce SB() TRCSpore(position, vigour, 0) EB();
}
for (int i = 0; i < VNT_INIT; i++) {
    position = V3f(-SIZE/2 + ran(SIZE), 0, -SIZE/2 + ran(SIZE));
    float vigour = getVigour(position, Ventarura);
    if (vigour > 0.25) nproduce SB() VNTSpore(position, vigour, 0) EB();
}
produce Generator();
}

Generator(): {
for (int i = 0; i < aglaophytonSpores.size(); i++) nproduce SB()
    AGLSpore(aglaophytonSpores[i], -1,
    10-10*(rarity[Aglaophyton]*nran(1.0,0.4))*delay[Aglaophyton]) EB();
aglaophytonSpores.clear();
for (int i = 0; i < asteroxylonSpores.size(); i++) nproduce SB()
    ASTSpore(asteroxylonSpores[i], -1,
    10-10*(rarity[Asteroxylon]*nran(1.0,0.4))*delay[Asteroxylon]) EB();
asteroxylonSpores.clear();
for (int i = 0; i < horneophytonSpores.size(); i++) nproduce SB()
    HRNSpore(horneophytonSpores[i], -1,
10-10*(rarity[Horneophyton]*nran(1.0,0.4)*delay[Horneophyton]) EB();

horneophytonSpores.clear();
for (int i = 0; i < nothiaSpores.size(); i++) nproduce SB()
    NTHSpore(nothiaSpores[i], -1,
           10-10*(rarity[Nothia]*nran(1.0,0.4)*delay[Nothia]) EB();

nothiaSpores.clear();
for (int i = 0; i < rhyniaSpores.size(); i++) nproduce SB()
    RHYSpore(rhyniaSpores[i], -1,
           10-10*(rarity[Rhynia]*nran(1.0,0.4)*delay[Rhynia]) EB();

rhyniaSpores.clear();
for (int i = 0; i < trichopherophytonSpores.size(); i++) nproduce SB()
    TRCSpore(trichopherophytonSpores[i], -1,
           10-10*(rarity[Trichopherophyton]*nran(1.0,0.4)*delay[Trichopherophyton])
           EB();

trichopherophytonSpores.clear();
for (int i = 0; i < ventaruraSpores.size(); i++) nproduce SB()
    VNTSpore(ventaruraSpores[i], -1,
           10-10*(rarity[Ventarura]*nran(1.0,0.4)*delay[Ventarura]) EB();

ventaruraSpores.clear();

produce Generator();
}

AGLSpore(position, vigour, delay): {
    if (vigour == -1) {
        vigour = getVigour(position, Aglaophyton);
        if (ran(1) > rarity[Aglaophyton] * sporeProductionMultipler[Aglaophyton])
            produce ;
        if (vigour < MIN_VIG + ran(RAN_VIG)) produce ;
    }
    if (delay > 0) produce AGLSpore(position, vigour, delay-1);
    printf("Aglaophyton spore germinated with vigour %f\n", vigour);
totalAGL++; currentAGL++;

addPlant(position);
produce SetColor(10-(int)(9*vigour)) SetWidth(0.01) RollR(ran(360))
    Base(0, position, vigour, nextID-1, Aglaophyton) StartGC() Down(90)
    Internode(0.05) AGLApex(0, false);
}

ASTSpore(position, vigour, delay): {
    if (vigour == -1) {
        vigour = getVigour(position, Asteroxylon);
       if (ran(1) > rarity[Asteroxylon] * sporeProductionMultiplier[Asteroxylon])
         produce;
        if (vigour < MIN_VIG + ran(RAN_VIG)) produce;
    }
    if (delay > 0) produce ASTSpore(position, vigour, delay-1);
    printf("Asteroxylon spore germinated with vigour %f\n", vigour);
    totalAST++; currentAST++;
}

addPlant(position);
produce SetColor(10-(int)(9*vigour)) SetWidth(0.02) RollR(ran(360))
    Base(0, position, vigour, nextID-1, Asteroxylon) StartGC() ASTApex(0, false);
}

HRNSpore(position, vigour, delay): {
    if (vigour == -1) {
        vigour = getVigour(position, Horneophyton);
       if (ran(1) > rarity[Horneophyton] *
            sporeProductionMultiplier[Horneophyton]) produce;
        if (vigour < MIN_VIG + ran(RAN_VIG)) produce;
    }
    if (delay > 0) produce HRNSpore(position, vigour, delay-1);
    printf("Horneophyton spore germinated with vigour %f\n", vigour);

totalHRN++; currentHRN++;

addPlant(position);
produce SetColor(10-(int)(9*vigour)) SetWidth(0.02) RollR(ran(360))
    Base(0, position, vigour, nextID-1, Horneophyton) StartGC() HRNApex(0, false);
}

NTHSpore(position, vigour, delay): {
    if (vigour == -1) {
        vigour = getVigour(position, Nothia);
        if (ran(1) > rarity[Nothia] * sporeProductionMultiplier[Nothia]) produce;
        if (vigour < MIN_VIG + ran(RAN_VIG)) produce;
    }
    if (delay > 0) produce NTHSpore(position, vigour, delay -1);
    printf("Nothia spore germinated with vigour %f\n", vigour);
    totalNTH++; currentNTH++;
}

RHYSpore(position, vigour, delay): {
    if (vigour == -1) {
        vigour = getVigour(position, Rhynia);
        if (ran(1) > rarity[Rhynia] * sporeProductionMultiplier[Rhynia]) produce;
        if (vigour < MIN_VIG + ran(RAN_VIG)) produce;
    }
    if (delay > 0) produce RHYSpore(position, vigour, delay -1);
    printf("Rhynia spore germinated with vigour %f\n", vigour);
    totalRHY++; currentRHY++;
}
addPlant(position);
produce SetColor(10-(int)(9*vigour)) SetWidth(0.02) RollR(ran(360))
    Base(0, position, vigour, nextID-1, Rhynia) StartGC() Down(90)
    RHYApex(0, false);
}

TRCSpore(position, vigour, delay): {
    if (vigour == -1) {
        vigour = getVigour(position, Trichopherophyton);
        if (ran(1) > rarity[Trichopherophyton] *
            sporeProductionMultiplier[Trichopherophyton]) produce ;
        if (vigour < MIN_VIG + ran(RAN_VIG)) produce ;
    }
    if (delay > 0) produce TRCSpore(position, vigour, delay-1);
    printf("Trichopherophyton spore germinated with vigour %f\n", vigour);
    totalTRC++; currentTRC++;
}

addPlant(position);
produce SetColor(10-(int)(9*vigour)) SetWidth(0.02) RollR(ran(360))
    Base(0, position, vigour, nextID-1, Trichopherophyton) StartGC()
    TRCApex(0, false);
}

VNTSpore(position, vigour, delay): {
    if (vigour == -1) {
        vigour = getVigour(position, Ventarura);
        if (ran(1) > rarity[Ventarura] * sporeProductionMultiplier[Ventarura])
            produce ;
        if (vigour < MIN_VIG + ran(RAN_VIG)) produce ;
    }
    if (delay > 0) produce VNTSpore(position, vigour, delay-1);
    printf("Ventarura spore germinated with vigour %f\n", vigour);
    totalVNT++; currentVNT++;
addPlant(position);
produce SetColor(10-(int)(9*vigour)) SetWidth(0.02) RollR(ran(360))
    Base(0, position, vigour, nextID-1, Ventarura) StartGC() VNTApex(0, false);
}

consider: Base;
Base(baseAge, pos, vigour, id, species) < AGLApex(order, sporified): {
    if (baseAge > DECAY) produce;
    else if (baseAge >= DEATH && !sporified) {
        for (int i = 0; i < AGL_NMSP; i++) {
            V3f sporePosition = V3f(pos.x+(ran(SPORE_RANGE*2)-SPORE_RANGE), 0, pos.z+(ran(SPORE_RANGE*2)-SPORE_RANGE));
            aglaophytonSpores.push_back(sporePosition);
        }
        produce AGLApex(order, true);
    }
    else if (baseAge >= DEATH) produce AGLApex(order, sporified);
    else if (ran(1) > (1.4-SVIG)*(order/3.0)) {
        float choice = ran(4);
        if (choice < 2) produce SB() Up(90) Internode(nran(0.12,0.012))
            RollR(nran(0,42)) SB() Right(30) Internode(nran(0.1,0.01))
            Sporangium() EB() SB() Left(30) Internode(nran(0.1,0.01)) Sporangium()
            EB() EB()
            SB() Right(nran(40,20)) Up(AGLANGLE) Internode(0.04) Down(AGLANGLE*2)
            Internode(0.04) Up(AGLANGLE) AGLApex(order+1, sporified) EB()
        SB() Left(nran(40,20)) Up(AGLANGLE) Internode(0.04) Down(AGLANGLE*2)
            Internode(0.04) Up(AGLANGLE) AGLApex(order+1, sporified) EB();
        else if (choice < 3) produce SB() Left(nran(30,20)) Up(AGLANGLE)
            Internode(0.04) Down(AGLANGLE*2) Internode(0.04) Up(AGLANGLE)
            AGLApex(order+1, sporified) EB();
        else produce SB() Right(nran(30,20)) Up(AGLANGLE) Internode(0.04)
```cpp
Down(AGLANGLE*2) Internode(0.04) Up(AGLANGLE) AGLApex(order+1, sporified) EB();
}
produce AGLApex(order, sporified);
}
Base(baseAge, pos, vigour, id, species) < ASTApex(order, sporified): {
  if (baseAge > DECAY) produce;
  else if (baseAge >= DEATH && !sporified) {
    for (int i = 0; i < AST_NMSP; i++) {
      V3f sporePosition = V3f(pos.x+(ran(SPORE_RANGE*2)-SPORE_RANGE), 0, pos.z+(ran(SPORE_RANGE*2)-SPORE_RANGE));
      asteroxylonSpores.push_back(sporePosition);
    }
    produce ASTApex(order, true);
  }
  else if (baseAge >= DEATH) produce ASTApex(order, sporified);
  else if (ran(0.8) > (1.4 - SVIG)*(order/2.0))
    produce RollR(68.8) SB() Left(45) ASTApex(order+1, sporified) EB()
    Internode(0.04*(1/(order+1.0))) ASTApex(order, sporified);
  produce ASTApex(order, sporified);
}
else if (baseAge >= DEATH) produce ASTApex(order, sporified);
else if (ran(0.8) > (1.4 - SVIG)*(order/2.0))
  produce RollR(68.8) SB() Left(45) ASTApex(order+1, sporified) EB();
  Internode(0.04*(1/(order+1.0))) ASTApex(order, sporified);
  produce ASTApex(order, sporified);
}
Base(baseAge, pos, vigour, id, species) < HRNApex(order, sporified): {
  if (baseAge > DECAY) produce;
  else if (baseAge >= DEATH && !sporified) {
    for (int i = 0; i < HRN_NMSP; i++) {
      V3f sporePosition = V3f(pos.x+(ran(SPORE_RANGE*2)-SPORE_RANGE), 0, pos.z+(ran(SPORE_RANGE*2)-SPORE_RANGE));
      horneophytonSpores.push_back(sporePosition);
    }
    produce HRNApex(order, true);
  }
  else if (baseAge >= DEATH) produce HRNApex(order, sporified);
}
```
else if (ran(1) > (1.4-SVIG)*(order/3.0)) {
    nproduce RollR(68.8);
    float choice = ran(4);
    if (choice < 2) produce SB() Right(nran(20,6))
        Internode(0.08*(1/(order+1.0))) NTHApex(order+1, sporified) EB() SB()
        Left(nran(20,6)) Internode(0.09*(1/sqrt((float)order)))
        NTHApex(order+1, sporified) EB();
    else if (choice < 3) produce SB() Left(nran(20,6))
        Internode(0.08*(1/(order+1.0))) NTHApex(order+1, sporified) EB();
    else produce SB() Right(nran(20,6)) Internode(0.08*(1/(order+1.0)))
        NTHApex(order+1, sporified) EB();
}
produce NTHApex(order, sporified);
}
Base(baseAge, pos, vigour, id, species) < NTHApex(order, sporified): {
    if (baseAge > DECAY) produce;
    else if (baseAge >= DEATH && !sporified) {
        for (int i = 0; i < NTH_NMSP; i++) {
            V3f sporePosition = V3f(pos.x+(ran(SPORE_RANGE*2)-SPORE_RANGE), 0,
                pos.z+(ran(SPORE_RANGE*2)-SPORE_RANGE));
            nothiaSpores.push_back(sporePosition);
        }
        produce NTHApex(order, true);
    }
    else if (baseAge >= DEATH) produce NTHApex(order, sporified);
    else if (ran(1) > (1.4-SVIG)*(order/3.0)) {
        nproduce RollR(68.8);
        float choice = ran(4);
        if (choice < 2) produce SB() Right(nran(20,6))
            Internode(0.08*(1/(order+1.0))) NTHApex(order+1, sporified) EB() SB()
            Left(nran(20,6)) Internode(0.09*(1/sqrt((float)order)))
            NTHApex(order+1, sporified) EB();
    }
else if (choice < 3) produce SB() Left(nran(20,6))
    Internode(0.08*(1/(order+1.0))) NTHApex(order+1, sporified) EB();
else produce SB() Right(nran(20,6)) Internode(0.08*(1/(order+1.0)))
    NTHApex(order+1, sporified) EB();
}
produce NTHApex(order, sporified);
}

Base(baseAge, pos, vigour, id, species) < RHYApex(order, sporified): {
    if (baseAge > DECAY) produce;
    else if (baseAge >= DEATH && !sporified) {
        for (int i = 0; i < RHY_NMSP; i++) {
            V3f sporePosition = V3f(pos.x+(ran(SPORE_RANGE*2)-SPORE_RANGE), 0,
                                    pos.z+(ran(SPORE_RANGE*2)-SPORE_RANGE));
            rhyniaSpores.push_back(sporePosition);
        }
        produce RHYApex(order, true);
    }
    else if (baseAge >= DEATH) produce RHYApex(order, sporified);
    else if (ran(1) > (1.4-SVIG)*(order/3.0)) {
        float choice = ran(4);
        if (choice < 2) produce SB() Up(90) Internode(nran(0.1,0.01))
            RollR(nran(0.42)) SB() Right(30) Internode(nran(0.06,0.006))
            Sporangium() EB() Internode(nran(0.08,0.008))
            SB() Left(30) Internode(nran(0.04,0.004)) Sporangium() EB() SetWidth(0)
            Internode(nran(0.05,0.005)) EB()
            SB() Right(nran(40,20)) Internode(nran(0.07,0.007)) RHYApex(order+1,
                sporified) EB() SB() Left(nran(40,20)) Internode(nran(0.07,0.007))
                RHYApex(order+1, sporified) EB();
        else if (choice < 3) produce SB() Left(nran(30,20)) Internode(0.07)
            RHYApex(order+1, sporified) EB();
        else produce SB() Right(nran(30,20)) Internode(0.07) RHYApex(order+1,
            sporified) EB();
    }
produce RHYApex (order, sporified);

Base (baseAge, pos, vigour, id, species) < TRCApex (order, sporified): {
  if (baseAge > DECAY) produce;
  else if (baseAge >= DEATH && !sporified) {
    for (int i = 0; i < TRC_NMSP; i++) {
      V3f sporePosition = V3f (pos.x+(ran(SPORE_RANGE*2)-SPORE_RANGE), 0,
                                pos.z+(ran(SPORE_RANGE*2)-SPORE_RANGE));
      trichopherophytonSpores.push_back(sporePosition);
    }
    produce TRCApex (order, true);
  } else if (baseAge >= DEATH) produce TRCApex (order, sporified);
  else if (ran(1) > (1.4 - SVIG)*(order/3.0)) {
    produce RollR(68.8);
    float choice = ran(4);
    if (choice < 2) produce SB() Right(nran(20,6))
      Internode(0.08*(1/(order+1.0))) TRCApex (order+1, sporified) EB() SB()
      Left(nran(20,6)) Internode(0.09*(1/sqrt((float)order)))
      TRCApex (order+1, sporified) EB();
    else if (choice < 3) produce SB() Left(nran(20,6))
      Internode(0.08*(1/(order+1.0))) TRCApex (order+1, sporified) EB();
    else produce SB() Right(nran(20,6)) Internode(0.08*(1/(order+1.0)))
      TRCApex (order+1, sporified) EB();
  }
  produce TRCApex (order, sporified);
}

Base (baseAge, pos, vigour, id, species) < VNTApex (order, sporified): {
  if (baseAge > DECAY) produce;
  else if (baseAge >= DEATH && !sporified) {
    for (int i = 0; i < VNT_NMSP; i++) {
    }
V3f sporePosition = V3f(pos.x+(ran(SPORE_RANGE*2)-SPORE_RANGE), 0, 
   pos.z+(ran(SPORE_RANGE*2)-SPORE_RANGE));
 ventaruraSpores.push_back(sporePosition);
}
produce VNTApex(order, true);
}
else if (baseAge >= DEATH) produce VNTApex(order, sporified);
else if (ran(1) > (1.4-SVIG)*(order/3.0)) {
   nproduce RollR(68.8);
   float choice = ran(4);
   if (choice < 2) produce SB() Right(nran(20,6))
       Internode(0.08*(1/(order+1.0))) VNTApex(order+1, sporified) EB() SB()
       Left(nran(20,6)) Internode(0.09*(1/sqrt((float) order)));
   VNTApex(order+1, sporified) EB();
else if (choice < 3) produce SB() Left(nran(20,6))
    Internode(0.08*(1/(order+1.0))) VNTApex(order+1, sporified) EB();
else produce SB() Right(nran(20,6)) Internode(0.08*(1/(order+1.0)))
   VNTApex(order+1, sporified) EB();
}
produce VNTApex(order, sporified);
}

// Cleanup
SB() EB(): { produce ; }
Base(baseAge, pos, vigour, id, species) < Internode(length): { if 
   (baseAge > DECAY) produce; }
Base(baseAge, pos, vigour, id, species) < Sporangium(): { if (baseAge > 
   DECAY) produce; }
Base(baseAge, pos, vigour, id, species) < Up(x): { if (baseAge > DECAY) 
   produce; }
Base(baseAge, pos, vigour, id, species) < Down(x): { if (baseAge > DECAY) 
   produce; }
Base(baseAge, pos, vigour, id, species) < Left(x): { if (baseAge > DECAY) produce; }
Base(baseAge, pos, vigour, id, species) < Right(x): { if (baseAge > DECAY) produce; }
Base(baseAge, pos, vigour, id, species) < RollR(x): { if (baseAge > DECAY) produce; }
Base(baseAge, pos, vigour, id, species) < SetColor(x): { if (baseAge > DECAY) produce; }
Base(baseAge, pos, vigour, id, species) < SetWidth(x): { if (baseAge > DECAY) produce; }
Base(baseAge, pos, vigour, id, species) < Elasticity(x): { if (baseAge > DECAY) produce; }
Base(baseAge, pos, vigour, id, species) < StartGC(): { if (baseAge > DECAY) produce; }

Base(age, pos, vigour, id, species): {
PlantInfo* nearest = nearestPlant(pos, id);
PlantInfo* thisPlant = getPlantInfo(id);

if (nearest != NULL) {
    float distance = sqrt((nearest->position.x - pos.x) * (nearest->position.x - pos.x) + (nearest->position.z - pos.z) * (nearest->position.z - pos.z));
    if (nearest->radius + thisPlant->radius > distance && thisPlant->radius < nearest->radius && thisPlant->alive) {
        age = DEATH + 1;
    }
}

if (thisPlant->alive && age >= DEATH) {
    thisPlant->alive = false;
}
if (age > DECAY + 1) {
if (species == Aglaophyton) currentAGL--;  
else if (species == Asteroxylon) currentAST--;  
else if (species == Horneophyton) currentHRN--;  
else if (species == Nothia) currentNTH--;  
else if (species == Rhynia) currentRHY--;  
else if (species == Trichopherophyton) currentTRC--;  
else if (species == Ventarura) currentVNT--;  
else printf("If you see this message, you done screwed up!\n");

deletePlant(id);
produce;
} else {
  if (thisPlant->alive) thisPlant->radius +=
    nran(0.012*vigour,0.004*vigour)*speciesRFactor[species];
produce Base(age+DT, pos, vigour, id, species);
}
}

interpretation:
maximum depth: 1;

Background(): {
nproduce SB() SetColor(15) MoveTo(0,0,-SIZE/2) RollR(180) Down(90)
CurrentTexture(0) TextureVCoeff(1/SIZE) CurrentContour(flat_CONTOUR)
SetWidth(SIZE) StartGC PointGC F(SIZE) PointGC() EndGC() EB();
produce;
}

AGLApex(order, sporified): {
  produce SetWidth(0) F(0.02) EndGC();
}

ASTApex(order, sporified): {
produce SetWidth(0) F(0.02) EndGC();
}

HRNApex(order, sporified): {
  produce EndGC() SetColor(12) Sphere(0.008);
}

NTHApex(order, sporified): {
  nproduce SetWidth(0) F(0.02) EndGC() f(-0.005) SetColor(13);
  for (int i = 0; i < 5; i++) nproduce SB() Right(90) f(0.004)
   Sphere(0.004) EB() RollR(137.5) f(-0.005);
  produce ;
}

RHYApex(order, sporified): {
  produce SetWidth(0) F(0.02) EndGC();
}

TRCApex(order, sporified): {
  nproduce F(0.005);
  for (int i = 0; i < 10; i++) nproduce Right(40) F(0.0042-i*0.0001);
  produce SetWidth(0) F(0.003) EndGC();
}

VNTApex(order, sporified): {
  nproduce SetWidth(0) F(0.02) EndGC() f(-0.005) SetColor(14);
  for (int i = 0; i < 6; i++) nproduce SB() Right(90) f(0.004)
   Sphere(0.004) EB() RollR(137.5) f(-0.005);
  produce ;
}

Internode(length): {
  produce F(length) PointGC();
}

Sporangium(): {
  produce EndGC() SetColor(12) Sphere(0.008);
Base(age, pos, vigour, id, species): {
    PlantInfo* temp = getPlantInfo(id);

    if (age >= DEATH) nproduce SetColor(11);
    produce MoveTo3f(pos) SB() f(0.001) Down(90) Circle(temp->radius) EB();
}