GENERATING CLIMBING PLANTS USING L-SYSTEMS

Johan Knutzen[1]*, Suguru Saito*[2]*, Masayuki Nakajima*[3]

Department of Computer Science
Graduate School of Information Science & Engineering
Tokyo Institute of Technology
W8-70 2-12-1 Ookayama Meguro-ku 152-8552, Japan
Email: {yohan[1], suguru[2], nakajima[3]}@img.cs.titech.ac.jp

ABSTRACT

We propose a novel method of procedurally generating climbing plants using L-systems. The goal of this research is to generate geometry for 3D-modelers, where procedurally generated content is used as a base for the final design.

The algorithm is fast and efficiently simulates external tropisms such as gravitropism and heliotropism, as well pseudo-tropisms. The structure of the generated climbing plants is discretized into strings of particles expressed using L-systems. The tips of the plant extend the branches by adding particles in its path, forming internodes.

A climbing heuristic has been developed that uses the environment as leverage when the plant is climbing, and effectively covers objects on which it grows. A fast method that sprouts leaves on the surface on which the plant is growing has also been developed, along with a heuristic that simulates the decrease in length, radius and leaf size.

Keywords: Climbing Plants, Oriented Particles, L-Systems, Procedural Content Creation

1. INTRODUCTION

This paper proposes a method of generating climbing plants procedurally using L-systems [7]. The L-system used is a graph grammar based language designed by Ole Kniemeyer [6]. This is an abstract language which extends L-systems with a graph rewriting formalism called Relational Growth Grammars [12]. The implementation of this language is called the XL programming language [5] and is a derivative of Java.

1.1 Motivation

One motivation behind this research is that in recent years content creation for computer games has become increasingly demanding. More content requires handcrafting by the designers, and much of the work is often repetitious.

One approach to minimize costs for the creation of game worlds regards reusing textures and geometries for objects that occur frequently throughout the game, such as trees and crates. Another approach is to generate recurring objects or even entire game worlds in a so-called procedural approach, yielding unexpected outcomes in form of appearance [2].

We would like to thank Noriaki Shinoyama for providing us with 3D models.

1.2 Climbing Plants

When describing the contents of a given scene in nature, the most obvious objects that could come to mind would be e.g. the trees, stones and houses. We perceive these as being the main contents of the scene, but just modeling these objects in a 3D-modeler would create a dull scene for rendering.

What makes nature beautiful in a sense, is its power to make each object look dynamic in synthesis. Climbing plants are excellent examples of a variety of plants which binds objects in a scene together. They try to dominate an area, striving for sunlight and coverage, with the aid of the environment. We argue that by adding these kinds of climbing plants, realism of a scene is increased.

2. PREVIOUS WORK

Benes and Millan [1] and Luft [8] use a particle system approach to grow climbing plants, where each branch is a string of particles. Another way of modeling climbing plants is to discretized the space into voxels and have the models grow from predefined geometric elements, according to rules based on intersection, proximity, and occlusion [3].

3. ALGORITHM

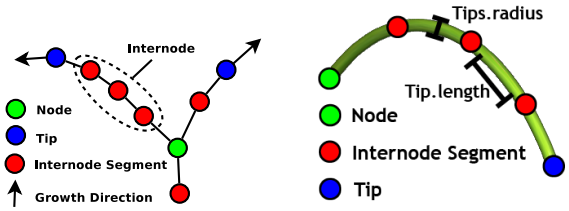
In order to generalize our system, emphasis is not put on strict definitions in the XL programming language. The system is instead abstracted to high-level L-system pseudo code, as the XL programming language is simply an implementation of an L-system.

3.1 Justification of Parameters

Patterns in nature arise as a result of evolution where plants strive for some optimum. If we study these patterns and map these as rules, we can effectively simulate plants. Lindenmayer saw this in his original work of simulating the

growth of algae [7]. In the case of climbing plants, we studied climbing plants in the wild and derived rules from our observations.

3.2 Growth of a Tip



(a) A tree with one internode split into two internodes; separated by a node.

(b) A tip with constant length and radius.

Fig. 1: The growth of a tip.

Consider fig. 1(a). A branch is discretized into a string of particles that are called internode segments. Each internode segment contains an affine transformation and a radius variable that represents its contribution to the curvature of the branch. The transformation M_i of an internode segment i is defined in eq. 1, and illustrated in fig. 1(b).

D : direction vector of the internode segment
N : normal vector of the internode segment
X : $D \times N$
P : translation of the internode segment

$$M_i = (X, N, D, P) \tag{1}$$

The P vector in eq. 1 is the length of the internode segment and in the ideal case of infinitesimal steps should be near zero in length.

The transformation from world-space to the local-space of the tip can be written as the product of the internode segments from the root of the tree to the tip, as shown in eq. 2.

$$T = \prod_{i=0} M_i \tag{2}$$

Moreover, since our L-System production rules are kept in a graph, all transformations can be found in $O(n)$ by traversing with e.g. depth-first. However, to increase performance, the transformation is cached in the Tip; resulting in an $O(1)$ lookup.

3.3 Production Rules

Consider the rules defined in program 1. A set of rules are applied to the tip at every time-step, resulting in a modification of the tip, and an extension of the internode. Each rule is applied every *rules[i].l* segments, with a probability of *rules[i].prob*. The basic rules of the tip can be defined as in table 1. The rules {*Tropism, Collision Avoidance, Decrease*} are applied at every time-step, and {*Die, Place Leaf*} at an empirically chosen frequency.

Program 1 Executing Production Rules

```
IF tip.alive()
   Tip -> segment(Tip) Tip
   Tip.segments += 1
   FOR each rules 0 ... i
   IF Tip.segments modulo rules[i].l = 0
      and probability(rules[i].prob)
   THEN
      rules[i].apply(Tip)
```

Table 1: L-System Production Rules

Tropism	Rotate the tip towards the tropisms.
Decrease	Decrease size of radius, length and leaf scalar.
Collision Avoidance	Avoid collision with the environment.
Die	Remove the tip.
Place Leaf	Place a leaf node.
Branch	Place a node, representing a splitting point.

3.4 Tropisms

Apart from adding internode segments in its path, the tip undergoes rotation due to tropisms. The strengths of the tropisms are chosen empirically, to meet the effect sought by the designer and to fit well with a given scene. Pseudo-tropisms can also be added; the effect of wind or just a vector that ensures that the plant grows towards a desired growth direction. For example, the designer might want the plant to skew across a wall, with less impact of gravity when growing up a wall, as if the plant was growing in a windy environment.

To rotate the tip towards a tropism direction, we first need to calculate the axis of rotation. With the world-space to local-space transformation from eq. 2, the axis of rotation in world-space can be evaluated as in eq. 3.

T^{-1} : local-space to world-space transformation
D : direction of the tip in local-space
a' : axis of rotation in local-space
a : axis of rotation in world-space
v : direction of the tip in world-space
d : direction of the tropism in world-space

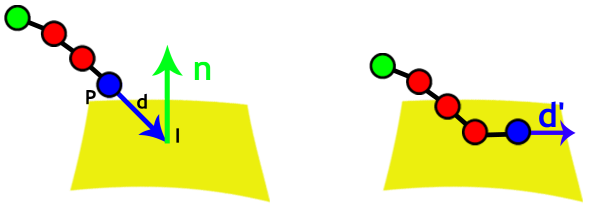
$$\begin{aligned} v &= T^{-1}D \\ a &= v \times d \\ a' &= Ta \end{aligned} \tag{3}$$

Now, using the axis of rotation in local-space a' from eq. 3, the new transformation for the tip can be calculated. This is evaluated by multiplying the transformation of the tip M_{tip} with the equivalent rotation matrix $R(a', \phi)$ of the rotation of ϕ degrees around a', as shown in eq. 4.

$$M'_{tip} = M_{tip}R(a', \phi) \tag{4}$$

3.5 Collision Avoidance

Benes and Millan [1] and Greene [3] use voxels for collision detection to check occupancy of a voxel by either the plant or some scene object. Our method uses ray-mesh intersection tests to query the environment for collisions and is different from previous work in that we use a BVH (Bounding Volume Hierarchy) instead of a uniform space subdivision scheme.



(a) Before the collision avoidance: the tip intersects the plane with a direction d.

(b) After the collision avoidance: the tip grows parallel with the plane with a new direction $\widehat{d'}$.

Fig. 2: Collision of a tip with a plane with normal n.

Consider fig. 2. When the tip encounters an intersection it is rotated so that its direction D is parallel with the plane of the intersected triangle, and so that the normal N is parallel with the normal n of the triangle.

This simple heuristic which is defined in eq. 5, ensures that the orthonormal vector X to N and D is parallel to the plane, and is used for sprouting leaves on the plane on which the tip is growing.

T : world-space to local-space transformation
I : intersection point in the plane
P : translation of the transformation of the tip
d' : new direction of the tip in world-space
d : direction of the tip in world-space

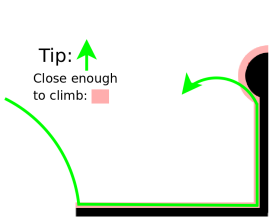
$$\begin{aligned} d' &= d - n(n \cdot d) \\ D' &= T\widehat{d'} \\ N' &= Tn \\ X' &= D' \times N' \\ P' &= I - T^{-1}P \end{aligned} \tag{5}$$

3.6 Climbing Heuristic

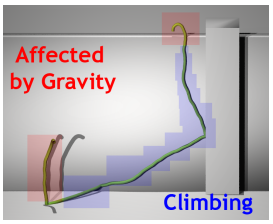
Consider fig. 3. A tip can have two states depending on its distance from the objects in the scene, either it is *climbing* or *not climbing*. If near enough, the tip ceases to be affected by gravity and instead starts to grow upwards, clinging to the nearest object in the scene.

t : tropism direction
s : tropism strength
$v = ts$: tropism with strength s
$\widehat{v}_{nc}$: total tropism direction when not climbing
$\widehat{v}_c$: total tropism direction when climbing

$$\widehat{v}_c = \frac{v_{gravity} + v_{light} + v_{random}}{||v_{gravity} + v_{light} + v_{random}||} \tag{6}$$



(a) When the tip is close to the pink area, it grows close on the black walls.



(b) The branch has been selected in blue when it is in a climbing state, otherwise pink.

Fig. 3: Climbing heuristic of the system, depicted in 2D and 3D.

$$\widehat{v}_{nc} = \frac{v_{up} + v_{light} + v_{adhesive} + v_{random}}{||v_{gravity} + v_{light} + v_{adhesive} + v_{random}||} \tag{7}$$

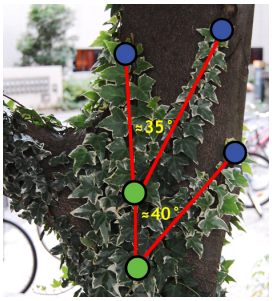
Consider eq. 6 and 7. These tropism vectors are used to compute the per time-step rotation the tip undergoes, in respective climbing state.

The nearest distance to an object in the environment is defined as the minimum distance between the tip and the barycentric coordinate of the tip, projected onto the plane of each triangle. If short enough (empirically chosen), tropism vectors $v_{adhesive}$, and v_{up} replace $v_{gravity}$, so that the tip is rotated upwards towards the nearest triangle.

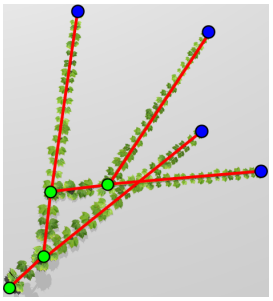
In either case, the final tropism of the tip is affected by a random tropism vector v_{random} and a heliotropism vector v_{light}. The randomness has been incorporated as a tropism to prevent artificial regularity, which deterministic L-systems suffer from [10].

The corresponding rotation matrix to the tropism vector $\widehat{v}_{nc}$ or $\widehat{v}_c$ is calculated, as described in section 3.4.

3.7 Branching Heuristic



(a) Branching of a Japanese Ivy at the Ookayama Campus of Tokyo Institute of Technology.



(b) Branching of a climbing plant, using our method.

Fig. 4: The branching of Japanese Ivy.

The branching angle of Japanese Ivy generally differ with circa $35\,° - 40\,°$ from the growth direction, as depicted in fig. 4(a). In the system, this is equivalent to placing a node at the splitting point, followed by a new tip. The node rotates the new tip away from the growth direction of the splitting tip, and results in two tips growing independently of each other. The result of this is depicted in fig. 4(b).

As with tropisms, we add randomness to the branching frequency in order to reduce regularity. The amount of randomness is proportional to how many nodes make up a branch and how much variance in the branching pattern the designer aims for.

3.8 Leaf Nodes

A leaf node signifies that leaves should be sprouted, and holds a scalar that is multiplied with the default size of a leaf. This scalar is set by the constructor of the leaf node and is retrieved from the tip.

3.9 Internode Segment Length, Radius and Leaf Size

Fig. 5: Above: Japanese Ivy in the wild. Below: An internode with an internode segment leaf spacing of of 2, decrease in leaf size per segment of 0.94, and a radius and length decrease of 0.97 per segment.

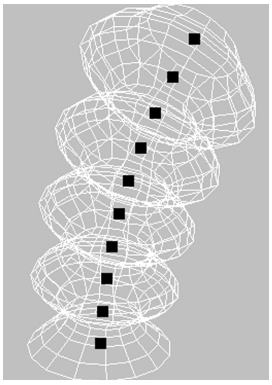
Consider our observation of Japanese Ivy in fig. 5. From this observation, we extracted the decrease in length, radius and leaf size between each internode segment. A general formula for how the radius, length and leaf size of branches is decreased can be written as in eq. 8.

$$
\begin{aligned}
a, b, c: &\quad \text{empirical constants.} \\
i: &\quad i\text{th segment in an internode.} \\
radius_{i+1} &= radius_i \times a \\
length_{i+1} &= length_i \times b \\
leafsize_{i+1} &= leafsize_i \times c
\end{aligned}
\tag{8}
$$

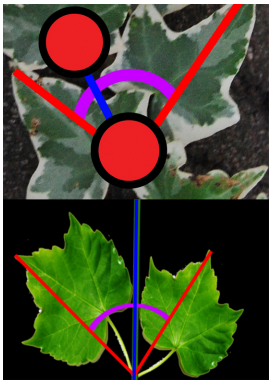
In our system we apply a rule which decreases these three variables at each time-step in the tip. A rendering with these relations is depicted in fig. 5.

3.10 Generating Geometry

The system consists of two procedures. The first one builds up a graph of productions with the XL programming language, until the production rules come to an end. By default the plant stops growing when a certain depth in the tree has been met, but the designer can also stop the production



(a) A cylindrical NURBS surface with varied radius. The black squares are vertices, with the same transformation as the internode segments.



(b) Above: An observation of the sprouting of leaves of Japanese Ivy. Below: Our sprouting heuristic, based on observations.

Fig. 6: Geometrical representation of the climbing plants.

manually. In the second procedure, the system traverses the nodes in the graph to create geometry.

Consider fig. 6(a). Branches are created, using cylindrical NURBS surfaces, where each internode segment signifies a vertex with a radius.

Leaves are created by sprouting two parallelograms at each leaf node. These parallelograms are sprouted parallel to the plane of the N axis of the leaf node, which is parallel with the plane on which the plant is growing. The parallelograms are also rotated from the growth direction, at which angles we have determined from observations, as illustrated in fig. 6.

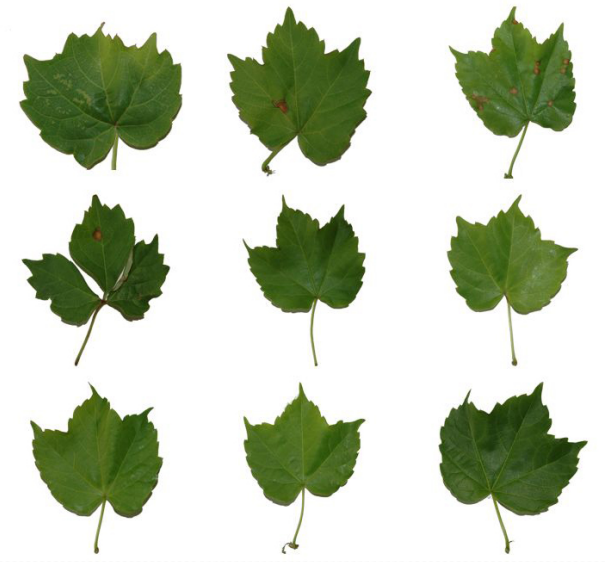


Fig. 7: Textures used for leaves. Original Photos taken at Ookayama Campus at Tokyo Institute of Technology.

The leaves are textured with random textures, chosen from the ones illustrated in fig. 7.

4. RESULTS

The first procedure to generate the productions, runs in $O(n(log(m)+m))$, where n equals the amount of internode segments added to the tree, and m equals to the amount of triangles. $O(log(m))$ is the time required to query the environment for intersections, and $O(m)$ is the time required to find the nearest triangle.

The second procedure which generates geometry takes $O(n)$, where n equals the amount of nodes in the production graph. The amount of tessellation for the branches is user-defined; an advantage from using NURBS surfaces.

Our method was implemented in Java, and can grow four plants in real-time with a model of 63,000 triangles, using seven internode segments per branch. The main bottleneck lies in the environment queries which scale with the amount of triangles. We experienced that this was not a big obstacle when working with larger models, since a low polygon model can be used as a substitution when generating the geometry.

It should also be noted that the spacing between internode segments need to be small enough, relative to the size of a triangle, in order to avoid unpredictable tip rotations.

To evaluate the realism of our method in contrast to another method, both methods must be applied to the same data. Unfortunately, we could not obtain an implementation of the work of Benes and Millan [1] and Greene [3], so we will focus on comparing our method to the one by Luft [8].

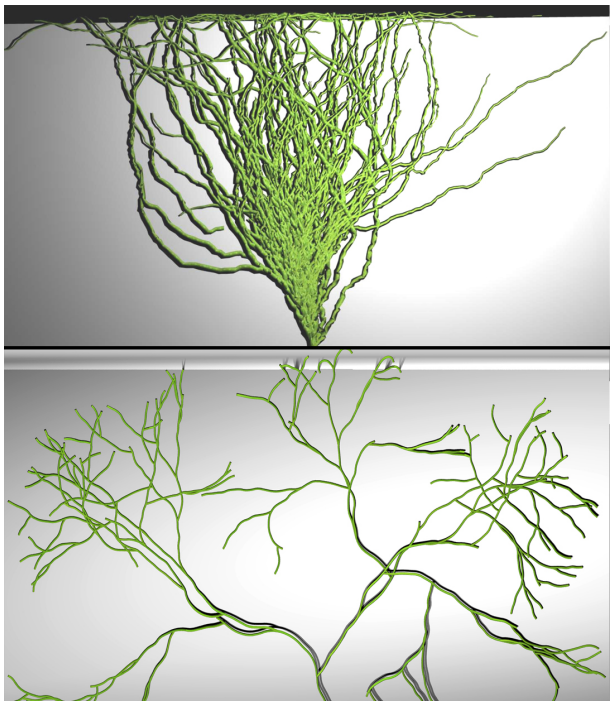


Fig. 8: Generated branches without leaves. Above: Luft's method. Below: Our method.

In our model we have implemented a branching heuristic, and the structural difference between ours and Luft's is illustrated in fig. 8. We believe that our method conforms better with our observation in fig 4, creating more self-similar structures.

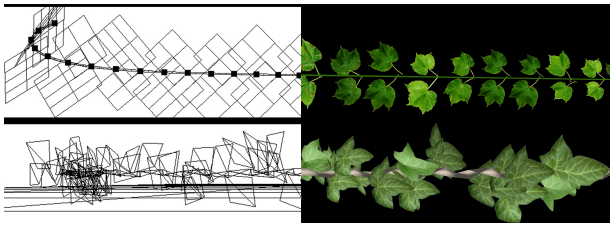


Fig. 9: A branch growing on a flat surface. Above: Luft's method. Below: Our method.

Our climbing heuristic is similar to that of Luft's method in that the plant grows close to the nearest triangle in the scene. However, we have improved upon the method by incorporating a leaf sprouting heuristic.

Consider fig. 9. Our method decreases the length and radius of each internode segment as well as the leaf size over time; conforming to our observations. Moreover, the effectiveness of our simple collision avoidance heuristic is demonstrated, along with the benefits of using NURBS surfaces instead of connected cylinders, as in Luft's implementation.

Our method benefits from using L-systems because we can create patterns by introducing production rules. Another benefit from using L-systems is that we can reproduce a set of productions, by simply saving the initial states and random seeds of the plants. Furthermore, since all tips are independent of each other, their production rules can be applied in parallel.

5. CONCLUSIONS

We have shown that simple heuristics can model collision avoidance and the effects of tropisms efficiently to generate geometry of climbing plants. Although biophysiologically inaccurate, our method is fast and easy to implement, with no reliance on infinitesimal steps to satisfy any numerical solver.

6. FUTURE WORK

When it comes to branches, a more clever way of generating the shape of the internodes comes close to mind. A backward propagating algorithm defining the radius of the internode segments could be developed, perhaps implementing the pipe-model as proposed by Shinozaki et. al [11]. The climbing heuristic of the plant can also be improved by swapping the closest triangle queries with a distance field algorithm, many of which are surveyed by Jones et. al [4].

The shape of leaves could also be improved, though at the expense of more complicated geometry. They could be modeled to bend, using e.g. NURBS surfaces. Other improvements include leaf fading, animation of leaves and branches as well as better sewing of the geometry where branches merge.

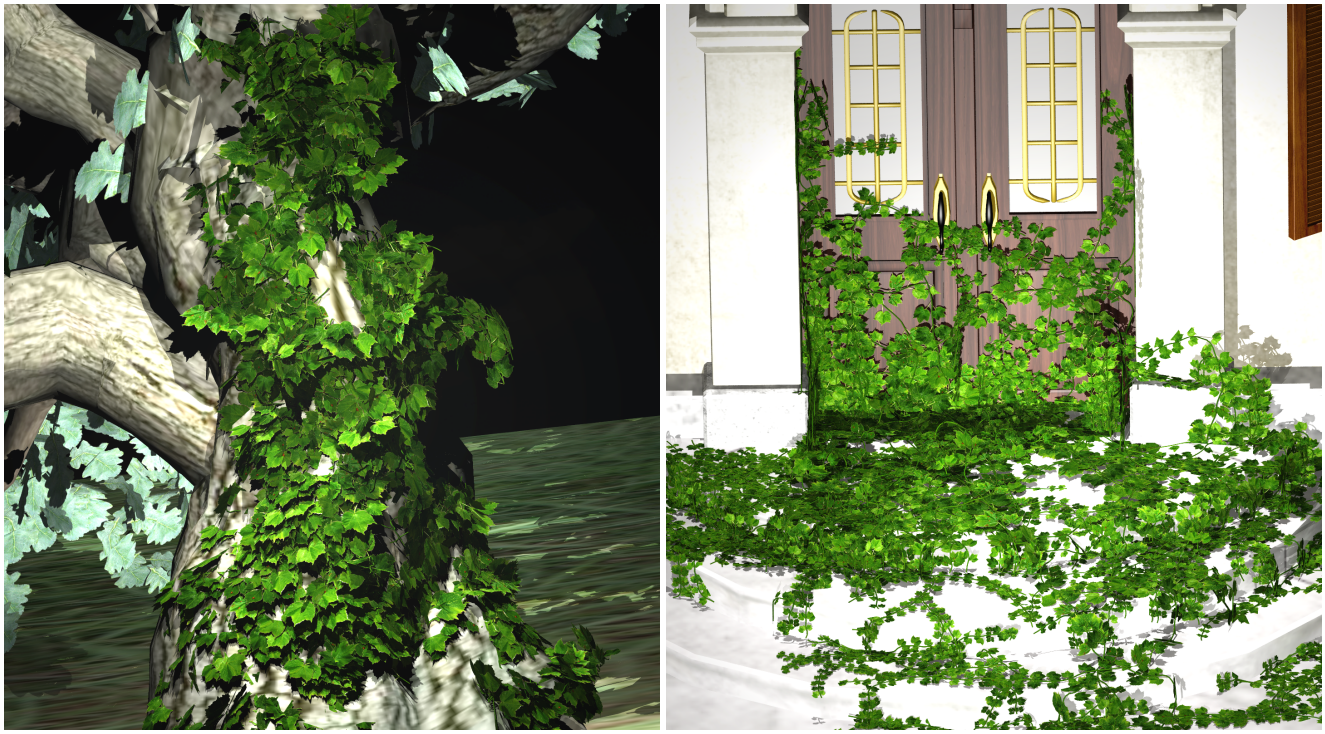


Fig. 10: Renderings of the climbing plants, ray-traced using PovRay [9].

7. REFERENCES

[1] Bedrich Benes and Erik Uriel Millan. Virtual climbing plants competing for space. *Computer Animation*, 2002.

[2] Greuter et al. Undiscovered worlds – towards a framework for real-time procedural world generation. In *Proceedings of the Fifth Intern. Digital Arts and Culture Conference*, 2003.

[3] N. Greene. Voxel space automata: modeling with stochastic growth processes in voxel space. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 175–184, New York, NY, USA, 1989. ACM.

[4] Mark W. Jones, J. Andreas Bærentzen, and Milos Sramek. 3d distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):581–599, 2006.

[5] Ole Kniemeyer. Rule-based modelling with the xl/groimp software. In Ulrike Brüggemann Harald Schaub, Frank Detje, editor, *The Logic of Artificial Life. Proceedings of 6th GWAL*, pages 56–65. Akademische Verlagsgesellschaft Berlin, 2004.

[6] Ole Kniemeyer. *Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling*. PhD thesis, BTU Cottbus, 2008.

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

[8] Thomas Luft. Ivy generator. `http://www.ivy-generator.com`, `Accessed 18 February, 2008.`

[9] Persistence of Vision Pty. Ltd. (2004). Persistence of vision raytracer (version 3.6). `http://www.povray.org/download/` , `Accessed 22 August, 2008.`

[10] P. Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[11] Kichiro SHINOZAKI, Kyoji YODA, Kazuo HOZUMI, and Tatuo KIRA. A quantitative analysis of plant form - the pipe model theory i. basic analysis. *Japanese Journal of Ecology*, 14(3):97–105, 19640601.

[12] Gerhard Buck-Sorlin Winfried Kurth, Ole Kniemeyer. Relational growth grammars - a graph rewriting approach to dynamical systems with a dynamical structure. In *Unconventional Programming Paradigms*, volume 3566/2005 of *Lecture Notes in Computer Science*, pages 56–72. Springer Berlin / Heidelberg, 2005.