Realism in Environment Sensitive Plant Models and their Animation
“I believe that producing pictures, as I do, is almost solely a question of wanting so very much to do it well.”

M.C. Escher
Acknowledgements

This dissertation would not have been possible without the contributions and support of many people. I consider myself fortunate to have worked with a number of truly remarkable people.

First and foremost, I wish to express my sincere gratitude to my supervisor prof. dr. Frank Van Reeth for the privilege of being a PhD student under his guidance and his confidence in my abilities. His broad knowledge on computer graphics subjects has been a great source of inspiration.

I have learned much from prof. dr. Fabian Di Fiore and dr. Tom Van Laerhoven. Being co-authors on some of my publications, they significantly contributed to the development of my research skills. Their creative input and vast expertise in computer animation techniques and non-photorealistic rendering greatly improved the quality of our publications.

Prof. dr. Eddy Flerackers, managing director of the EDM, prof. dr. Philippe Bekaert, prof. dr. Karin Coninx, prof. dr. Wim Lamotte, prof. dr. Kris Layten, prof. dr. Chris Raymaekers and Peter Vandoren were also from invaluable support. Furthermore, I would like to thank the members of the EDM secretariat, Ingrid Konings and Roger Claes, for all the things they arranged for me during the past six years and for their administrative assistance.

I enjoyed working with all my other colleagues, who also became good friends throughout the years. I especially want to thank Bert De Decker, Jeroen Dierckx, Mark Gerrits, Bjorn Geuns, Tom Haber, Erik Hubo, Tom
Jehaes, Pieter Jorissen, Tom Mertens, Peter Quax and Cedric Vanaken.

I also appreciate the countless times I could rely on the many other friends I made during my studies. For the many hours we shared talking, laughing, partying and having the best times of our life, I would like to mention the following people: Maarten Alders and Robrecht Reyskens, Roel Guldentops, Tine Bories, Michael (Wichem, The Pipetman, etc.) Van Loock, Mieke Pessemier, Nick Verhaert, Kim Deliège, Koen (de Witters) Witters, and Leen Vander Kuylen,

Finally, I thank my parents, Jos and Paulette, my sister Vera, and particularly my truly remarkable wife Annelies, for their support, love and encouragement during the occasional times of hard work.

Diepenbeek, May 2007

Dedicated to my parents,
for their never-ending support and care
Abstract

For computer related visualization purposes, the beauty of plants and trees plays an important role in the representation of the world we live in. Many techniques exist to model and to animate them, but often these result in an “unnatural” look. Especially representing the structural complexity of an aged tree in a convincing way, in many cases still remains a challenging task to perform. For these reasons, many modelers/animators prefer to use traditional methods, which are very labor-intensive and require a good knowledge of the underlying phenomena involved. In this dissertation we present two main contributions to facilitate this cumbersome task.

First, we describe two algorithms to incorporate the influence of the environment into the development of plant models and/or virtual trees. The first method, using an L-System based plant representation, heuristically measures the amount of light that reaches different plant parts and modifies the plant structure accordingly. As a result the shape of the resulting model can reflect the presence of light sources, nearby objects and other neighboring plants easily.

A second algorithm mathematically improves this technique by estimating the illumination density and its mean incident direction near growing points. Again, specific growing behavior is synthesized in a convincing way. We demonstrate the usability of both methods on several different species of plants and trees.
The second part of this dissertation focusses on the animation aspect. By exploiting the realism inherently present in video data of plants and by simulating the laws of their motions in a physically-based manner, we propose techniques to create new perpetual animations of plants and trees.

First, we extend the video textures method. By exploiting the symmetry in plant motions the playback of a sequence of video frames is reversed, yielding new perpetual animations of these plants. Next, we describe a method to create endless, user controlled cartoon animations. As a final contribution, a geometry-based technique is presented to animate three dimensional tree models. After acquiring motion data (video-based or physically-based), the set of motion samples is extended using techniques from computer-assisted keyframe animation. By organizing the full set of motions into a motion graph, and performing graph walks on it in a goal based manner, the animator is given the opportunity to create new, endless tree animations, interactively, fast and without much effort. The resulting animations are much broader than their inputs and still reflect the realistic motions captured earlier.
CONTENTS

2.2 Contributions ................................................. 10
2.3 Aiming for Structural Realism ................................ 11
  2.3.1 Biological Aspects .................................. 11
2.4 Automated Plant Modeling .................................. 14
  2.4.1 Local-to-global Methods ............................ 15
  2.4.2 Global-to-local Methods ............................ 22
  2.4.3 Modeling Large Natural Scenes .................. 25
2.5 Plant Visualization ......................................... 30
2.6 Concluding Remarks ....................................... 31

3 The Competition for Light and Space ..................... 33
  3.1 Introduction and Motivation ............................ 33
  3.2 Approach ................................................ 34
  3.3 The Supported Interactions ............................. 35
    3.3.1 Object interaction ................................ 35
    3.3.2 Light Interaction ................................ 37
    3.3.3 Neighbor Interaction ............................. 38
    3.3.4 Light vs. Neighbor influence .................. 42
  3.4 Discussion ............................................... 43

4 Plant Growth based on a Ray Density Estimation ........ 45
  4.1 Introduction and Motivation ............................ 45
  4.2 Approach ................................................ 46
  4.3 Representing the Environment Illumination ............ 47
    4.3.1 A Dynamic Process ................................ 49
  4.4 Querying the Environment .............................. 50
  4.5 Examples of Supported Phenomena ..................... 52
    4.5.1 Branch Bending ................................... 54
    4.5.2 Leaf and Flower Orientation ..................... 56
    4.5.3 Branch, Leaf and Flower Size ................... 57
    4.5.4 Complex Behavior ................................ 58
  4.6 Results .................................................. 58
  4.7 Discussion ............................................... 65

II Plant Animation .............................................. 67

5 Video Textures of Plants .................................... 69
  5.1 Introduction and Motivation ............................ 69
  5.2 Our Contributions ...................................... 70
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6.2</td>
<td>Providing a Physically-based Support System</td>
<td>111</td>
</tr>
<tr>
<td>7.6.3</td>
<td>Capturing the Animation Trajectory</td>
<td>112</td>
</tr>
<tr>
<td>7.7</td>
<td>Motion Processing and Optimization</td>
<td>113</td>
</tr>
<tr>
<td>7.7.1</td>
<td>Constructing the Basic Animation Path</td>
<td>113</td>
</tr>
<tr>
<td>7.7.2</td>
<td>Towards a Motion Graph</td>
<td>114</td>
</tr>
<tr>
<td>7.7.3</td>
<td>Optimizing the Motion Graph</td>
<td>117</td>
</tr>
<tr>
<td>7.8</td>
<td>Motion Synthesis</td>
<td>117</td>
</tr>
<tr>
<td>7.8.1</td>
<td>Generating New Animations</td>
<td>117</td>
</tr>
<tr>
<td>7.9</td>
<td>Results</td>
<td>118</td>
</tr>
<tr>
<td>7.10</td>
<td>Discussion</td>
<td>120</td>
</tr>
<tr>
<td>8</td>
<td>Concluding Remarks</td>
<td>123</td>
</tr>
<tr>
<td>8.1</td>
<td>Summary and Future Directions</td>
<td>123</td>
</tr>
<tr>
<td>Appendices</td>
<td></td>
<td>129</td>
</tr>
<tr>
<td>A</td>
<td>L-System-based Plant Growth</td>
<td>129</td>
</tr>
<tr>
<td>A.1</td>
<td>Defining L-Systems</td>
<td>129</td>
</tr>
<tr>
<td>A.2</td>
<td>Geometric Interpretations of Strings</td>
<td>130</td>
</tr>
<tr>
<td>A.3</td>
<td>L-System Taxonomy</td>
<td>132</td>
</tr>
<tr>
<td>A.3.1</td>
<td>D0L-Systems</td>
<td>132</td>
</tr>
<tr>
<td>A.3.2</td>
<td>Bracketed L-Systems</td>
<td>132</td>
</tr>
<tr>
<td>A.3.3</td>
<td>Nondeterministic L-Systems</td>
<td>133</td>
</tr>
<tr>
<td>A.3.4</td>
<td>Stochastic L-Systems</td>
<td>133</td>
</tr>
<tr>
<td>A.3.5</td>
<td>Context Sensitive L-Systems</td>
<td>134</td>
</tr>
<tr>
<td>A.3.6</td>
<td>Parametric L-Systems</td>
<td>134</td>
</tr>
<tr>
<td>A.3.7</td>
<td>Open L-Systems</td>
<td>135</td>
</tr>
<tr>
<td>A.4</td>
<td>Plant Growth based on L-Systems: a Simplified Pipeline</td>
<td>136</td>
</tr>
<tr>
<td>B</td>
<td>Scientific Contributions and Publications</td>
<td>139</td>
</tr>
<tr>
<td>B.1</td>
<td>Topic Related Publications</td>
<td>139</td>
</tr>
<tr>
<td>B.2</td>
<td>Off-topic Publications</td>
<td>140</td>
</tr>
<tr>
<td>C</td>
<td>Samenvatting (Dutch Summary)</td>
<td>141</td>
</tr>
<tr>
<td>C.1</td>
<td>Introductie</td>
<td>141</td>
</tr>
<tr>
<td>C.2</td>
<td>Deel I: Plant Modeling</td>
<td>142</td>
</tr>
<tr>
<td>C.2.1</td>
<td>Realisme in de Vorm van Planten</td>
<td>142</td>
</tr>
<tr>
<td>C.2.2</td>
<td>De Competitie voor Licht en Ruimte: een Eenvoudige Methode</td>
<td>144</td>
</tr>
</tbody>
</table>
C.2.3 Plantengroei Gebaseerd op een Schatting van de Lichtdensiteit

C.3 Deel II: Plant Animation

C.3.1 Videotexturen van Plantenbewegingen

C.3.2 Cartoon Animaties Gebaseerd op het Video Textuur Principe

C.3.3 Boomanimaties Afgeleid van Fysisch Gebaseerde Simulaties

C.4 Algemeen Besluit

Bibliography
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Different types of realism: two examples</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Three aspects of environment sensitivity</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Simple tree drawings created by Aono and Kunii</td>
<td>16</td>
</tr>
<tr>
<td>2.3</td>
<td>Constrained plant growth based on the use of voxel space automata</td>
<td>17</td>
</tr>
<tr>
<td>2.4</td>
<td>An example of a sculptured plant based on L-Systems developed by Prusinkiewicz</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>Trees competing for light based on the algorithms by Mech et al.</td>
<td>19</td>
</tr>
<tr>
<td>2.6</td>
<td>The use of positional information to model a Pellaea Falcata (Sickle Fern) leaf</td>
<td>21</td>
</tr>
<tr>
<td>2.7</td>
<td>Results from the work by Lintermann and Deussen</td>
<td>23</td>
</tr>
<tr>
<td>2.8</td>
<td>The use of multiset L-Systems to distribute 4 different plant species</td>
<td>26</td>
</tr>
<tr>
<td>2.9</td>
<td>A scene rendered using a hybrid level-of-detail approach</td>
<td>27</td>
</tr>
<tr>
<td>2.10</td>
<td>State of the art results from work done by Alsweis and Deussen</td>
<td>29</td>
</tr>
<tr>
<td>2.11</td>
<td>Non-photorealistic visualizations of skeleton-based tree models</td>
<td>30</td>
</tr>
<tr>
<td>3.1</td>
<td>Example of a stochastic, parametric, context sensitive, open L-System</td>
<td>35</td>
</tr>
<tr>
<td>3.2</td>
<td>A synthetic plant shape based on an enclosing primitive</td>
<td>36</td>
</tr>
<tr>
<td>3.3</td>
<td>Trees interacting with objects in their surrounding environment</td>
<td>36</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.4</td>
<td>An octree as a solution to the problem of branches competing for space</td>
<td>37</td>
</tr>
<tr>
<td>3.5</td>
<td>A plant growing in an environment with indirect illumination</td>
<td>39</td>
</tr>
<tr>
<td>3.6</td>
<td>Rays preventing trees from growing in each other’s shadow by exchanging information about their positions</td>
<td>40</td>
</tr>
<tr>
<td>3.7</td>
<td>4 trees competing for space</td>
<td>41</td>
</tr>
<tr>
<td>3.8</td>
<td>4 trees competing for space: top view</td>
<td>41</td>
</tr>
<tr>
<td>3.9</td>
<td>Light attraction and neighbor repulsion</td>
<td>42</td>
</tr>
<tr>
<td>3.10</td>
<td>Skeletons of alternative tree species generated using an environment sensitive L-System</td>
<td>43</td>
</tr>
<tr>
<td>4.1</td>
<td>Storing references to nearby light rays in an octree</td>
<td>48</td>
</tr>
<tr>
<td>4.2</td>
<td>An adaptive octree depth based on a user-specified search distance</td>
<td>52</td>
</tr>
<tr>
<td>4.3</td>
<td>Three incremental refinements of the nearest rays around a query point</td>
<td>53</td>
</tr>
<tr>
<td>4.4</td>
<td>The schematic representation of a branch bending towards the dominant illumination direction</td>
<td>56</td>
</tr>
<tr>
<td>4.5</td>
<td>Several plants, created from the same L-System, with different environmental properties</td>
<td>59</td>
</tr>
<tr>
<td>4.6</td>
<td>A similar result, but based on a different L-System</td>
<td>60</td>
</tr>
<tr>
<td>4.7</td>
<td>Plant models generated by applying the ray density estimation technique</td>
<td>62</td>
</tr>
<tr>
<td>4.8</td>
<td>Another set of plants generated using our ray density estimation technique</td>
<td>63</td>
</tr>
<tr>
<td>4.9</td>
<td>Snapshots of an animation depicting the growth of the plant under the table from Figure 4.7</td>
<td>64</td>
</tr>
<tr>
<td>5.1</td>
<td>Reversing video playback at a symmetric frame position</td>
<td>72</td>
</tr>
<tr>
<td>5.2</td>
<td>A graphical representation of a simple clock sequence</td>
<td>75</td>
</tr>
<tr>
<td>5.3</td>
<td>Still from the original video sequence of a swinging pendulum</td>
<td>78</td>
</tr>
<tr>
<td>5.4</td>
<td>Examples of possible subjects for new video synthesis</td>
<td>79</td>
</tr>
<tr>
<td>6.1</td>
<td>The pipeline of our process to create endless or repetitive animations</td>
<td>86</td>
</tr>
<tr>
<td>6.2</td>
<td>Two extreme poses of a flower and an in-between frame generated from them</td>
<td>90</td>
</tr>
<tr>
<td>6.3</td>
<td>Two extreme poses of a human face and an in-between frame generated from them</td>
<td>90</td>
</tr>
<tr>
<td>6.4</td>
<td>The graphical interpretation of the cost matrix C</td>
<td>94</td>
</tr>
<tr>
<td>6.5</td>
<td>The 5 initial keyframes of a simple flower animation</td>
<td>99</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.6</td>
<td>Several snapshots from the flower animation</td>
<td>99</td>
</tr>
<tr>
<td>6.7</td>
<td>Five keyframes used to create a moving figure animation</td>
<td>99</td>
</tr>
<tr>
<td>6.8</td>
<td>Several snapshots from the moving figure cartoon animation</td>
<td>99</td>
</tr>
<tr>
<td>6.9</td>
<td>5 images displaying different placements of the mouth</td>
<td>100</td>
</tr>
<tr>
<td>6.10</td>
<td>Several snapshots from the speech animation</td>
<td>100</td>
</tr>
<tr>
<td>6.11</td>
<td>5 keyframes used to generate a facial expression animation</td>
<td>100</td>
</tr>
<tr>
<td>6.12</td>
<td>Several snapshots from the facial expression animation</td>
<td>100</td>
</tr>
<tr>
<td>7.1</td>
<td>A schematic overview of the 3 phases of the data path</td>
<td>110</td>
</tr>
<tr>
<td>7.2</td>
<td>A tree skeleton with a mass-spring system attached to it</td>
<td>113</td>
</tr>
<tr>
<td>7.3</td>
<td>The initial motion backbone</td>
<td>114</td>
</tr>
<tr>
<td>7.4</td>
<td>The creation of a transition when two path segments cross</td>
<td>115</td>
</tr>
<tr>
<td>7.5</td>
<td>A closed directed motion graph, derived from the motion backbone</td>
<td>116</td>
</tr>
<tr>
<td>7.6</td>
<td>Snapshots from three animations, constructed from the same tree model</td>
<td>119</td>
</tr>
<tr>
<td>7.7</td>
<td>Two examples of the applicability of our technique</td>
<td>120</td>
</tr>
<tr>
<td>7.8</td>
<td>100 tree skeletons, fully animated at real-time frame rates</td>
<td>121</td>
</tr>
<tr>
<td>A.1</td>
<td>A schematic representation of the Turtle’s state</td>
<td>131</td>
</tr>
<tr>
<td>A.2</td>
<td>A fractal drawn using Turtle Interpretation</td>
<td>132</td>
</tr>
<tr>
<td>A.3</td>
<td>The main steps encountered during the processing of an environment sensitive L-System</td>
<td>136</td>
</tr>
</tbody>
</table>
LIST OF FIGURES
List of Tables

4.1 A method to process the growth of an environment sensitive plant | 50
4.2 L-System 1, illustrating how branch bending is achieved | 55
4.3 A possible result from the processing of L-System 1 | 55
4.4 L-System 2, capable of interacting with the environment | 57
4.5 L-System 3, combining the techniques used in L-System 1 and 2 | 58
6.1 The algorithm to optimize the initialized cost matrix $C$ | 95
6.2 An interactive approach to the random animation generation method | 97
CHAPTER 1

Introduction

1.1 Realism in Plant Modeling and Animation

During the last decades, hardware performance has improved significantly and the knowledge involved in software development has reached new heights. Both are becoming more and more an indispensable asset to our daily life. Hence, people are confronted more often with artificially created data. Games, for example, consist in many cases of virtual locations where people spend a lot of their spare time.

These technological improvements also boosted the development and availability of the Internet, revealing its possibilities to almost anyone, and providing an easy way for people to stumble upon new virtual content. Furthermore, artificial data is also encountered in large quantities in the world of television and movies, both depending heavily on the creation of plausible graphical effects. The creation of cartoon movies or short cartoon animations for example are based on many hours of labor by many different artists.

It is important to notice how, for example when viewing such drawn animations or virtual environments, the content of these scenes often is limited
to its most important components. In several cases this relates to performance issues (e.g., games), but can also be dependent on production costs or time constraints. As a result, backgrounds or additional scenery are neglected completely, represented only partially or visualized very unrealistically. Even though much effort is put in the visualization of the main topics, the absence of a complete virtual representation can influence the realism perceived by the viewer which is often required to fully appreciate the scene.

Artificial natural environments play an important role in this context. Vegetation is all around us; we are confronted with it on a daily basis. Hence, we have developed a very good impression on how the world around us works, how things should or shouldn’t look or how they should behave. Violations against this cognitive model strike us immediately and disturb our sense of realism. Even though we are not always aware of how nature works, we easily take for granted how plants require sufficient amounts of light and nutrition to grow, how they obey the law of gravity, and how they are moved by the same wind we feel. Ignoring such simple phenomena can have an important impact on the credibility of such a virtual scene.

The type and degree of realism supported, depends on the purpose of the scene depicted. In the case of a cartoon animation a full support for realism is not always required (hence the “cartoon” effect), but often contributes to the quality of the production. Figure 1.1.a depicts a still from Madagascar, showing how on one hand much effort was put in acquiring convincing plant shapes, but on the other hand, they were visualized in a more amusing cartoon style. In virtual environment applications or feature films, however, aiming for realistic behavior and a convincing appearance of the depicted content is often a straightforward requirement. Figure 1.1.b illustrates how in the production of The Lord of The Rings: The Two Towers a different kind of realism was pursued and much research was performed to render trees photorealistically [Aitken 03]. These rendering techniques were, however, applied to “living” tree characters who required two stems to be able to walk. Hence, less structural realism was aimed for.

1.2 Thesis

Synthesizing realistically looking natural scenes remains a challenging and important problem to be solved with computer graphics research. This challenge stems not only from the complexity and diversity of objects and natural
Figure 1.1: Different types of realism: (a.) a still from the animation film *Madagascar*, depicting realistically shaped trees and plants, rendered in an amusing cartoon style; (b.) *up*: a still from *The Lord of the Rings: The Two Towers*, displaying the visual realism of the tree models used. *down*: production art illustrating how a tradeoff was made between local and global structural realism (e.g., fine realistic details vs. the addition of a second stem), to allow for convincing, human-like tree shapes.

phenomena interacting together, but also from the huge amount of small details that should be modeled to obtain plausible simulations and a satisfying amount of realism in the resulting models.

A need still exists for simple, intuitive, and interactive modeling and simulation techniques capable of handling the different aspects of such synthetic sceneries. Consequently, the purpose of our research was to explore novel techniques for modeling, animating and simulating some of the natural phenomena involved in the development of plants and tree, relieving the modeler/animator partially from these cumbersome tasks.
Hence, we formulate the thesis of this dissertation using the following two statements:

*By simulating the environment sensitive process involved in real-life plant growth, the perceived realism of artificial plants and trees can be increased.*

*For their animation, both the employment of existing video data and the simulation of the underlying physical laws of motion serve as a strong basis from which new, credible animations can be synthesized in a simple, fast and interactive way.*

After summarizing our main contributions, we address both of these topics in more detail throughout the remainder of this document.

### 1.3 Contributions

We would like to attract the attention to the diversity in the use of the word “realism”. For plants, like for most objects, many different aspects contribute to this term. This dissertation, however, focusses on our research concerning a few specific aspects of the structural realism of plants and trees, and how they can be animated in a simple and convincing way and not on the photorealistic properties of their visualizations.

Concerning this structural realism, we investigated phototropism, a phenomena relating to the environment sensitivity of plants and influencing their shape drastically. For animation purposes, we contribute to the creation of plausible, perpetual animations of plants and trees based on existing video data or physically-based simulated motions. Both research topics are described into more detail below.

#### 1.3.1 Environment Sensitive Plant Models

We start by presenting two algorithms that allow for the simulation of plant growth in a realistic way, while taking into account environmental illumination, spatial occupancy and the nearby presence of other plants.

By augmenting the L-System, plant growth is modeled using a mechanism for determining the predominant illumination directions by analyzing the distribution, density and direction of incident light hitting plant parts. Several photomorphogenetic responses are triggered, resulting in visually more realistic plant shapes, indicating clearly their environment sensitivity by bending branches, adapting their growth rate, orienting their leaves and flowers, etc.
1.3.2 Video-based or Physically-based Plant Animations

We describe an alternative to the traditional method for creating video textures, bearing in mind the video-based animation of plants. By exploiting the symmetric properties of the frames in a given video sequence, a new endless video stream is generated by reversing video playback when appropriate. Hence, we aim at a larger set of possible input video's for video texture creation.

Based on insights from this research, the use of computer-assisted keyframe animation techniques, and the concept of motion graph walks, we provide a method that facilitates the creation of cartoon animations. From a small amount of input data, we allow animators to easily and interactively create vector-based or image-based perpetual animations.

Next, we aim for realistic animations of tree skeletons. After creating intermediate data bridging the different instances from an acquired set of physically-based simulated motion samples, the animator is presented with a simple and interactive method to generate new, plausible tree animations. These captured motions can be exploited to represent up to 100 fully animated, skeleton-based tree models, in real-time, even without the use of algorithms that accelerate the visualization of large object quantities.

1.4 Document Overview

After this introduction to the topic of our dissertation, the text is organized into two parts:

- Part I: Plant Modeling, focussing on realism in the plant shape by considering responses to environmental influences.

- Part II: Plant Animations, addressing realism in short plant animations, using video-based data and extending physically-based acquired motions.

The first part provides a detailed insight into our contributions to the field of realistic plant modeling. Following up a short introduction (Chapter 2) on the concept of environment sensitive plant models, Chapter 3 reveals our first attempt to achieve realistic responses by artificial plant models to the environmental illumination, based on a simple, yet efficient method. Next, Chapter 4 demonstrates a more sophisticated algorithm producing environment sensitive plant models based on an illumination density estimation.

The second part of this dissertation covers our contributions to plant animation. In Chapter 5 we describe an extension to the video textures concept,
allowing for simple, video-based plant animations. Chapter 6 covers an implementation of a fast and simple technique to create perpetual cartoon animations from small inputs, based on computer-assisted keyframe animation. Finally, Chapter 7 demonstrates how interactive tree animations are made by extending physically-based acquired motion data, using simple interpolation techniques, and performing graph walks on the new data utilizing well chosen data transitions.

We end this dissertation by describing our overall conclusions and directions for future work (see Chapter 8).
Part I

Plant Modeling
CHAPTER 2

Realism in the Plant Shape

2.1 Introduction and Motivation

Plant modeling has always been a challenging task and can be traced back to the early sixties, when Ulam applied cellular automata to simulate the development of branching patterns, thought of as an abstract representation of plants [Ulam 62].

Due to the complex geometry involved, not only the modeling process itself is labor-intensive, also the interaction of the model with its environment yields several important problems that need to be solved.

Historically, much of the effort in computer graphics has been directed towards rendering precisely defined geometrical shapes. The complexity of most objects is generally low enough to specify their geometrical properties in terms of a simplified approximation.

Natural objects, on the other hand, offer a more profound challenge. Due to their large structural complexity (e.g., in terms of topology), approximating them with simpler variants often results in a loss of realism. A natural scene may contain hundreds of trees, thousands of plants and billions of grass
blades, rocks, pebbles, etc. But even a single tree may easily be characterized by hundreds of thousands of leaves and thousands of branches, branchlets, and stems oriented in different directions. The unimaginable large number of polygons required to define each part clearly invokes the demand for more automated methods able to create different kinds of plant models and the many variations derived from them.

During the last decades many researchers have incrementally improved different approaches accomplishing this, varying from basic fractal-based approaches to the use of fully developed plant modeling tools, commercially available to anyone.

In an early stage, related research was mainly focussed on the actual creation of plausible plant models and only simple heuristics were put in practice to represent the environment sensitive character of these natural objects. During the last few years, however, more effort was put into the search for better algorithms taking into account the environment in which the plants grow, and allowing for the creation of more realistic and thus better accepted plant models.

2.2 Contributions

The research presented in the first half of this dissertation contributes to the latter part of this limited history of plant modeling and mainly focusses on the environment sensitivity of the models and an increase to the structural realism of the plants created.

By determining the amount and the direction of incident illumination at those positions in the plant models where changes in the topology or elongation takes place, the further development of the model can be directed to take into account environmental information. Furthermore, the presence of other plants or trees (often blocking a large amount of sunlight) is taken into consideration. As a result, the models of the virtual plants created by our methods clearly demonstrate their struggle in the competition for light and space, a phenomenon easily observable in real life.

Our goal is to end up with a basic description of a realistic model in terms of positions, ages, etc. Shaded, textured and illuminated visual representations of these models can be added afterwards to satisfy the individual needs of interested users, but they are not considered here.
2.3 Aiming for Structural Realism

We already mentioned how structural realism plays an important role in the credibility of a plant model. To this extent, the environment sensitive character of a plant model needs to be considered.

2.3.1 Biological Aspects

Studies in the field of photomorphogenesis\textsuperscript{1} revealed the existence of several very interesting kinds of plant behavior. In this dissertation we focus on three different interactions between plants and their surroundings:

- light interaction;
- object interaction;
- neighbor interaction.

**Light interaction.** In their book, Kendrick and Kronenberg state: ‘The perception of light direction yields important information enabling organisms to optimize their position in the natural environment by appropriate orientation movements’ [Kendrik 86]. A nice example of this is observed from trees, trying to spread their leaves in such a way that the majority of them receive as much of the available light as possible (because certain components of the light spectrum are necessary for photosynthesis\textsuperscript{2}).

   This phenomenon is called *phototropism*: growth movements of plants in relation to the light direction. A tree provides a solution to this need by obtaining a good distribution of its branches and by growing in the direction that receives most of the light (to reach for even more light). Similar behavior is also particularly visible with sprouts of new plants. Figure 2.1.a shows a typical positive phototropism observed on Mung Bean seedling, subjected to uni-directional illumination.

**Object interaction.** It is a well known phenomenon how plant development is influenced by the presence of objects. Many different plant species even require the presence of other plants or objects to attach to. Ivy, for example, is a common known plant species that grabs onto other plants or objects. So

\textsuperscript{1} the branch of biology studying the interaction of trees with light.

\textsuperscript{2} a chemical reaction that provides energy to the tree.
does Bindweed, which is shown in Figure 2.1.b. In both cases, the resulting shape of the plant is influenced by the shape of the object it attaches to.

On the other hand, the development of the plant is often restricted, due to the presence of nearby objects. When growing inside a small room or corner, for instance, the branches of the plant cannot elongate beyond the walls and thus are forced to bend towards empty spaced locations.

**Neighbor interaction.** Kendrick and Kronenberg also describe how trees “sense” each other’s presence within a certain neighborhood, sometimes even when there is a large distance between them. Plants try to avoid each other by growing in opposite directions so they would not end up in each other’s shadow. The adaptation described results from a biological reaction by the leaves in which they only absorb certain wavelength components of the light spectrum and reflect others, which suggests how neighbor interaction can be seen as a special case of light interaction.

When a tree receives a lot of light from a certain direction, containing only those components of the light spectrum not used by the plant and supposed to be reflected, and just a few rays that can be absorbed (the so called photobiological-active portion of the radiant spectrum), a system will be triggered inside the plant to grow away from that direction. This means that the wavelength of the light that is transported by the light beams functions as some sort of information exchange between adjacent trees. The same rules apply for reflections from walls or other objects, because they also filter certain components of the light spectrum. The effect is, however, strongest for light reflected by a tree part (a leaf, a branch), because the specifically required wavelengths will be missing in their reflected light beams.

Figure 2.1.c illustrates this phenomenon. The image displays several trees growing close to each other. The shape of the right-hand side tree (containing more leaves) is clearly influenced by the presence of other nearby trees, even without actual collisions. The branch and leaf density resulting from the presence of the larger left-hand side tree and the smaller trees in the middle forces the right-hand side tree to further develop his canopy in the open area, which is available on the right side of the image.

**Other phenomena.** Many other plant related responses can be observed from nature. Examples are gravitropism (a turning or growth movement by a plant or fungus in response to gravity) and heliotropism (the diurnal motion of plant parts – flowers or leaves – in response to the direction of the sun). Our techniques, presented in the upcoming chapters, can easily be extended...
Figure 2.1: The three aspects of environment sensitivity addressed in this dissertation: (a.) Light Interaction (Mung Bean seedling); (b.) Object Interaction (Bindweed); (c.) Neighbor Interaction.
to take into account such phenomena, but for demonstration purposes, we restricted our research to the three responses mentioned above.

During many years, several techniques have been proposed to model plants and trees that take into account the responses to their environment. In the following sections we will investigate the many different approaches taken by other researchers in their pursuit for structural realism in plant modeling.

2.4 Automated Plant Modeling

Lane et al. described how, in general, two main approaches exist for the creation of virtual natural scenes, differentiated by the way they approach the problem statement [Lane 02]: many researchers focus on local-to-global methods and others work on global-to-local methods. Boudon et al. demonstrated how this classification is also applicable on individual plant models [Boudon 03].

In the local-to-global models, the user characterizes individual components (modules) of a plant, and the modeling algorithm integrates these components into a complete structure. This approach is particularly useful in the modeling and simulation of plant development for biological purposes. However, controlling the overall plant form remains difficult.

On the contrary, with the global-to-local models, the user characterizes global aspects of the plant shape, such as its overall silhouette and the density of branch distribution. These modeling algorithms employ this information to infer details of the plant structure. The global-to-local approach provides a more direct and intuitive control of visually important aspects of plant form, and therefore is preferable in applications where visual output is of primary importance.

Both approaches focus on the creation of a structural model, positioning the different components (like branches, twigs, etc.) from which a plant is constructed.

In contrast, other methods to visualize plants and trees in a convincing way have been examined, which do not aim for the construction of such a topological model.

Image-based and video-based approaches, for example, try to represent vegetation by means of well chosen slices of image data. These slices give, when combined or interpreted, the effect of a realistic plant shape [Max 99, Qin 03, Behrendt 05, Colditz 05, Quan 06]. Such techniques have proven themselves
useful when visualizing large scenes with large amounts of vegetation, but the resulting models often lose their credibility when viewed from a closer distance and even more when animation is added.

In this document, we will limit our overview on related work to the earlier mentioned local-to-global and global-to-local methodologies because of their relation to our research, and we will focus on the most important contributions made to them.

2.4.1 Local-to-global Methods

In order to describe growth of living organisms, Lindenmayer [Lindenmayer 68] introduced the notion of a parallel rewriting system as a formalism for simulating the development of multicellular organisms. The Lindenmayer systems, or L-Systems, attracted the interest of many researchers, and their theory was soon extensively developed [Lindenmayer 72, Rozenberg 73, Lindenmayer 76]. Currently, natural phenomena such as growth, death, reproduction, and information flow in growing plants are easily modeled via L-Systems. Readers unfamiliar with the concept of L-Systems can find a brief introduction to this topic in Appendix A.

Although a geometrical interpretation of strings was at the origin of L-Systems, they were not applied to picture generation until 1984, when Aono and Kunii, and Smith used them to create images of trees and plants. Examples are shown in Figure 2.2.

Aono et al. presented botanical trees as models of botanical objects, first by defining their developmental rules in a discrete grammatical form, then by defining them in continuous geometric forms [Aono 84]. As a result, more complex branching patterns could be described. Moreover, for the first time, interaction with the environment was simulated. Based on four basic geometric models, representations of simple tree models were acquired, taking into account wind, sunlight and gravity using uniform and/or non-uniform deviations.

Even though Aono and Kunii included coarse environmental factors, detailed awareness of the environment geometry during the growth process was not taken into account.

Smith too was one of the first authors describing fractals and formal plant descriptions for computer graphics [Smith 84]. He introduced a deterministic grammar-based modeling approach called graftals, which could model plants and trees. Graftals are fractal-like in their ability to create forms of great
complexity from simple instructions, yet are more versatile in comparison to fractals that are strictly self-similar.

Also, the attention was attracted to the phenomenon of data-base amplification, or the possibility of generating complex structures from compact data sets, which is inherent in L-Systems and forms the cornerstone of L-System applications to image synthesis.

A different approach was taken by De Reffye et al., who presented a model for the growth of plants and trees incorporating botanical knowledge about their architecture. In the late 80’s, the strength of their method was the ability to produce a larger set of different tree models, ranging from weeping willows to cedar trees, frangipani trees, poplars, pine trees, wild cherry trees and even herbs. All these models were based on the same procedural methods, controlled by the birth and death of growing buds. Another very important benefit from the model was the integration of time, enabling to view the aging of a tree and to study the incidence of factors such as insect attacks, the use of fertilizers, planting densities, etc. Their technique mainly focussed on the usability for agronomy and botany purposes and less towards explicit visual realism.

Arvo and Kirk derived the form of a plant from the paths of one or more environment-sensitive automata. Their heuristic method, based on the querying of the environment with raycasting, provided a very nice first
2.4 Automated Plant Modeling

Figure 2.3: Constrained plant growth based on the use of voxel space automata: (a.) a voxel space around a cylindrical object; (b.) regular plant growth; (c.) restricted plant growth using a proximity constraint; (d.) the combination of a proximity constraint and a helically twisting constraint; (e.) a result of Greene’s voxel based plant growth simulation, displaying a high degree of object interaction (Image’s source: [Greene 89]).

An attempt to take into account criteria such as the proximity of objects and the amount of illumination available to the plant.

In 1989 the term Voxel Space Automata was introduced to describe growth processes that sensed and reacted to a voxel environment [Greene 89]. Models were “grown” from predefined geometric elements according to rules based on simple relationships like intersection, proximity, and occlusion. From any point in voxel space the size, shape, and proximity of neighboring objects could be determined by inspecting the records of nearby voxels. Growth could be controlled with simple biases and constraints. Examples of this technique are visualized in Figure 2.3.

Inspired by the work of Arvo and Kirk, the technique presented by Greene et al. efficiently estimated the illumination at each plant “node” at each growth iteration by casting rays into the voxel environment – where ray casting was interpreted as tiling a ray in voxel space – allowing the simulation of a reaction to light, including heliotropism.

When familiar with the research on plant modeling, the name Prusinkiewicz should ring a bell. He himself and other members of his research lab are known for their significant contributions to the research in plant modeling.
and animation. Their first contributions to the subject [Prusinkiewicz 86, Prusinkiewicz 88] augmented the concept of L-Systems to visualize fractal curves, ranging from Koch curves, to classic space-filling curves, but also pictures of plants and trees.

In the early nineties, they introduced a mathematical framework for modeling plants and the simulation of their development in a manner suitable for animation [Prusinkiewicz 93]. The key concept was the integration of discrete and continuous aspects of model behavior into a single formalism, called differential L-Systems (dL-Systems). These systems form a continuous-time extension to parametric L-Systems, where L-System-style productions express qualitative changes to the model (for example, the initiation of a new branch), and differential equations capture continuous processes, such as the gradual elongation of internodes.

A few months later, Prusinkiewicz et al. described another extension to the use of Lindenmayer systems in a manner suitable to mimic the interaction between a developing plant and its environment [Prusinkiewicz 94]. The new formalism was illustrated by modeling the response of trees to pruning, yielding synthetic images of sculptured plants, as they are often found in topiary gardens (see Figure 2.4). The results from their work were further extended [Prusinkiewicz 95], allowing the propagation of information through the plant model, the simulation of the development of plants attacked by insects and the pruning of branches that provide an insufficient amount of resources to the plant.

Meanwhile, Hammel et al. presented a methodology for constructing developmental plant models incorporating biological data, and illustrated it using a model of green ash [Hammel 95]. Their approach consisted of defining a qualitative plant model expressed as a skeletal L-System guiding the acquisition and statistical analysis of field data. After estimating specific growth parameters, a quantitative model could be defined, expressed as a differential L-System.

In 1996, Mech et al. introduced a modeling framework capable of simulating and visualizing a wide range of interactions at the level of plant architecture [Mech 96, Prusinkiewicz 96a]. This framework extended the formalism of Lindenmayer systems with constructions needed to model bi-directional information exchange between plants and their environment. They illustrated the proposed framework with models and simulations that capture the development of tree branches limited by collisions, the colonizing growth of clonal plants competing for space in favorable areas, the interaction between roots
2.4 Automated Plant Modeling

Figure 2.4: An example of a sculptured plant, based on L-Systems developed by Prusinkiewicz capable of sensing the environment and hence, automatically pruned to a spiral shape (Image source: [Prusinkiewicz 94]).

Figure 2.5: Trees competing for light based on the algorithms by Mech et al.: (up) in the position of growth; (down) moved apart revealing the adaptation of crown geometry to the presence of the neighbor tree (Image source: [Mech 96]).
competing for water in the soil, and the competition within and between trees for access to light. Results from their technique are shown in Figure 2.5.

In the same year, Prusinkiewicz et al. drew the attention to the ease by which a mathematical methodology like L-Systems could produce results unfaithful to actual observations made in nature. In response, *Subapical Bracketed L-Systems* were presented [Prusinkiewicz 96b], restricting the creation of new branches to the apices (i.e. tips) of the existing branches and not just anywhere on the plant model.

Benes presented in 1996 a method for the estimation of the amount of light reaching a plant [Benes 96] and presented an improvement to it one year later [Benes 97]. His method was based on the capturing of several snapshots of the plant from different sampling positions on the celestial hemisphere. By representing each leaf with a different color and evaluating the histograms of the captured images, the amount of light exposure from all different angles could be estimated for each leaf. This information was used to simulate plant growth taking into account environmental illumination and was able to mimic growth around obstacles. However, the method was computational expensive and several issues remained unaddressed (e.g., indirect illumination, aliasing problems during image capturing, etc.)

In “Visual Models of Plant Development”, Prusinkiewicz, Hammel, Hanan and Měch surveyed the applications of L-Systems to the modeling of plants [Prusinkiewicz 97], with an emphasis on the results obtained since the comprehensive presentation of this area in “The Algorithmic Beauty of Plants” [Prusinkiewicz 90]. At that time, one of the most important new developments was the introduction of programming constructs that enhanced the use of L-Systems to a language for the construction of developmental algorithms to be used as inputs for simulation programs.

As an example of such a plant modeling related programming construct, an L-System-based modeling language was described as a component of *cpfg* [Prusinkiewicz 99], a program used to simulate and visualize plant development [Han 92, Měch 97, Měch 98, Prusinkiewicz 00b].

At the conceptual level, it improved the design, specification, documentation and comparison of different models. At the level of model implementation, it allowed to develop the graphical capabilities needed to visualize these

---

3the successor of *pfg: plant and fractal generator* [Han 88, Prusinkiewicz 89], with continuous parameters
2.4 Automated Plant Modeling

Figure 2.6: The use of positional information to model a Pellaea Falcata (Sickle Fern) leaf. The image shows how the leaflet and internode length, the branching angle and the stem shape are modeled easily using simple graph modifications (Image source: [Prusinkiewicz 01]).

models, without the need to reimplement them every single time. Also, the language facilitated interactive experimentation with plant models.

Power presented, in collaboration with Prusinkiewicz [Power 99], a system for the interactive manipulation of plants modeled using L-Systems. Their application allowed the user to locally deform a plant model while preserving its botanical accuracy. The approach focussed on the interaction with the L-System model at the level of the L-string.

Based on cpfg, Jirasek et al. described a model that made it possible to visually capture the shape of branches resulting from the combined effect of weight, heuristic tropisms, and the contact of plant parts with each other and with obstacles in the environment [Jirasek 00]. The incorporation of biomechanics into L-Systems made it possible to explore different branching architectures relatively easily.

Prusinkiewicz et al. also explored the idea of plant modeling with functions that relate features of a plant to their positions along the different plant axes [Prusinkiewicz 01]. Their procedural approach allowed for the easy capturing and controlling of the visually important aspects of plant appearance, like posture, the arrangement of components, and the overall silhouette. Furthermore, their technique also removed the tedium of specifying and placing each plant component individually. The presented algorithms worked faster and were
demonstrated by recreating the form of several plants found in nature. The structures modeled using this technique range from individual leaves to compound herbaceous plants and trees. Figure 2.6 demonstrates how positional information was used to model a Sickle Fern leaf.

In 2002, a method was described by Beneš and Millan, allowing climbing plants to compete for space using oriented particles, able to sense the environment [Beneš 02b]. Using directed random walks and traumatic reiteration their research allowed plants to find the best way to grow.

2.4.2 Global-to-local Methods

In contrast to the local-to-global methods, the goal of a global-to-local method is to define a convincing plant shape with a certain amount of detail and allow the underlying algorithms to fill in the blanks. A correct visual appearance of the plant is aimed for and less attention is drawn to the underlying structure.

The earlier mentioned work done by Beneš and Millan was based on the attempts by Reeves and Blau to model natural objects with controllable structured particle systems [Reeves 83], [Reeves 85]. It required the user to specify geometrical constraints (e.g., the overall silhouette) for a tree and allowed a procedural method to generate the composition of the tree by adding sub-branches recursively. This technique was also used to model larger natural scenes containing several plant models and/or grass.

Weber and Penn were also influenced by these techniques and put special emphasis on modeling the overall structure of the tree and not on the underlying botanical principles [Weber 95]. Specifying the shape geometrically and restricting the model to grow within the bounds of that shape accomplished this. Their model was able to handle random parameters so that a very large number of structural variations could be generated from the design specifications of a particular tree species.

When applying a local-to-global approach, controlling the global aspects of the overall plant shape directly sometimes proves difficult. For instance, when using an L-System based method, users receive no direct feedback because they must rerun the simulation after changing the parameters.

In response, Lintermann and Deussen decided to describe plant models by an object hierarchy [Lintermann 96], [Deussen 97], [Lintermann 99], where components encapsulate both geometrical and structural data together with
2.4 Automated Plant Modeling

Figure 2.7: Results from the work by Lintermann and Deussen: (a.) a plant modeled by composing a structural graph (depicted on the right) from components representing the different components of the plant shape and their properties; (b.) a Weeping Willow subjected to gravity; (c.) a Philodendron growing around a stick. (Image’s source: [Deussen 97])

the corresponding methods to visualize them. The geometric properties were handled as parameters of these components and a structure tree was composed by combining icons (graphical representations of the components). In the end, each tree represented a set of creation rules which defined a context-free rule system. From these techniques an interactive plant modeling program was developed, named Xfrog [Greenworks 98].

By combining the power of a rule-based approach with the intuitiveness of generic tree modeling methods, a general technique was presented that could generate nearly all kinds of plants by using one consistent description. Furthermore, the user was able to model the geometry by standard interaction techniques like editing splines or using free form deformations, and received instant feedback of his modifications. Result from the method are depicted in Figure 2.7.

This component-based method allowed for the modeling of several simple tropisms. Gravitropism was simulated while creating a Weeping Willow, and a cylindrical tropism for modeling a Philodendron around a stick could be achieved easily. Both examples are depicted in Figure 2.7b and 2.7c. The influence of wind was simulated by a horizontal tropism and a plant could be forced to grow along a wall. More complex environment sensing (e.g., deriving
locally the direction of incident direct and indirect illumination), however, was not fully supported, but in stead mimicked using well chosen heuristics or techniques like free form deformation.

Boudon et al. addressed the paradigm where providing a small set of parameters to compose a plant model results in easier user interactions but higher similarities on the different levels of the model because of the insufficiency to specify plant characteristics in detail, whereas the availability of a larger set of controllable parameters allows for finer detail and more distinct features on the different levels of the plant model. This, however, requires the user to assemble a large amount of well-defined settings even when creating a very simple model [Boudon 03]. A large number of parameters is often tedious to manipulate and difficult to comprehend.

Consequently, they provided a method to manage the different parameters involved in plant model manipulation by introducing decomposition graphs as multiscale representations of plant structures and presented interactive tools for tree design that operate on decomposition graphs.

In 2004, Galbraith et al. described a global-to-local technique based on the inverse modeling paradigm [Galbraith 04]. The user defined model parameters like the overall lengths and orientations of branches or functions indicating their possible loss. From this data trees were constructed by means of an improved use of implicit surfaces and new blending techniques, both able to better represent smaller details like bud scale scars and branch bark ridges.

Recently, Rudnick et al. presented a new technique for plant modeling [Rudnick 07], based on similar work done by Linsen et al. [Linsen 05] and focussing on real-time visualizations. Their tree growth model was based on global functions of time that described global properties of the tree during growth (e.g., tree height, diameter at breast height, and crown height and width). These functions were derived from real measured data of seven tree species and mapped to local production rules. By simultaneously and iteratively applying such local production rules to all branches, trees were produced with a natural branching structure, fitting global user defined properties.

They also included the influence of light on the growth of the tree, using a data structure that supported a fast, heuristic computation of the available light at every position inside the crown at any time. In the end, their GPU-based implementation allowed for real-time plant visualization and animation.
2.4 Automated Plant Modeling

2.4.3 Modeling Large Natural Scenes

This far, we mainly focussed on techniques to model individual plant models. Based on different plant characteristics controlled by the user, a model is grown or modeled as a composition of hundreds or thousands of structures like branches or leaves.

This methodology can, however, also be applied to a higher level of natural object modeling. Globally, the user determines how species react to each other and how much light and space they require for further development and procedural methods simulate the combined growth processes of all the elements in the scene, taking into account these predefined constraints. During simulation, local conflicts are encountered and resolved, resulting in the virtual competition of plants for light, nutrition, space, etc., reflecting the interactions of plants with each other and with their environment.

Correct models of these ecosystems have a wide range of existing and potential applications, including computer-assisted park and garden design, games, and the prediction and visualization of external effects on landscapes.

In order to render such natural scenes realistically, terrain must be modeled and plants need to be distributed in a realistic manner. Geometric models of individual plants, consistent with their positions within the ecosystem, must be synthesized to populate the scene. Due to the large amount of environmental interaction encountered during their simulation, they clearly make up an important part of this dissertation.

As early as in 1984, Gardner proposed a strategy for creating visually complex natural scenes [Gardner 84]. The terrain and the trees were modeled using a relatively small number of geometric primitives (quadric surfaces). Their perceived complexity resulted from procedural textures controlling the color and the transparency of tree crowns.

Like we mentioned in Section 2.4.2 Reeves and Blau made a different attempt to natural scene modeling. They synthesized larger scenes using (structured) particle systems, with the order of one million particles per tree. Their approach required several approximations, since detailed information about the environment was not available.

Weber and Penn introduced a heuristic multi-resolution representation specific to trees [Weber 95], which allowed to reduce the number of geometric primitives in the models that occupy only a small portion on the screen.
Deussen et al. developed a system to address the major different tasks involved in the creation of large complex landscapes [Deussen 98]. They decomposed the process of image synthesis into different stages: modeling the terrain, specifying the plant distribution, modeling the individual plants, and rendering the entire scene. Each of these stages operates at a different level of abstraction, and provides a relatively high-level input for the next stage.

After terrain editing, the plant distribution was determined by hand, by ecosystem simulation, or by a combination of both techniques. Given parameterized procedural models of individual plants, the geometric complexity of the scene was reduced by approximate instancing, in which similar plants, groups of plants, or plant organs were replaced by instances of representative objects before the scene was rendered. By employing these procedural models in all stages of the modeling pipeline and using object instancing as the primary paradigm for reducing the size of the geometric representation of the rendered scenes, collections of up to 100,000 detailed plant models could be visualized, but required several hours of processing and rendering time.

Lane and Prusinkiewicz formalized and extended the methods for defining plant distributions originally proposed by Deussen [Lane 02] and clearly distinguished between a local-to-global approach, in which the distribution of plant densities was determined by a simulation of interactions between the individual plants and a global-to-local approach, in which positions of individual plants were inferred from given large-scale density distributions.

For the local-to-global method, an extension to the L-System formalism called multiset L-Systems was introduced, capable of modeling groups
2.4 Automated Plant Modeling

Figure 2.9: A scene rendered using a hybrid level-of-detail approach. The combination of polygons, points and lines enabled to better convey the shape of different object categories in more distant views (Image source: [Deussen 02]).

of plants, rather than single plants alone. These L-Systems were applied to simulate the essence of self-thinning, succession, and clustering of plants in ecosystems. Figure 2.8 illustrates schematically the resulting distribution of four different plant species and a view on the corresponding virtual forest scene.

Meanwhile, Deussen et al. presented a new system for interactively rendering large outdoor scenes [Deussen 02]. They introduced a method based on a good pre-processing of the model data, to render the scene using a point- and line-based level-of-detail approach, optimized using object hierarchies. The combined polygon/point and line representation enabled to better convey the shape of different object categories in distant views. Based on the idea that best visual results are achieved when objects are not uniformly reduced, a method of importance reduction allowed for the representation of visual important parts of a scene in a higher quality than others. Hence, drastically reducing the unimportant parts of the scene while supporting a good visual impression. An example of such a hybrid representation is given in Figure 2.9.

In 2004, Beneš presented an approach for modeling plant competition by means of clusters [Benes 04]. This technique was influenced by his earlier work on ecosystem modeling [Benes 02a] presenting a simple local-to-global method which remained stable, even when sudden, drastic changes were made to the
Realism in the Plant Shape

environment.

Instead of considering a plant as a solitary element competing for its space and life with other plants, in his new work Beneš clustered groups of plants of the same specie with a boundary that was formed by an implicit curve. Plant competition occurred at two different levels – within one plant species and between different types of plants. When two plants of the same species met, their ecological neighborhoods were combined; when plants of a different type met, their neighborhoods repelled. The resulting plant distributions looked realistic. However, they were not based on biological principles, but rather on well chosen heuristics.

Recently, Deussen and Lintermann published a book on their research [Deussen 05], illustrating and exemplifying methods for the creation of artificial plant models, and the application of these methods within areas such as simulation, virtual reality, botany, landscaping, and architecture.

Some researchers, for instance Desbenoit et al., worked on a different aspect of the realistic perception of natural scenes and focussed on the addition of finer detail to the scene like the growth of mushrooms and lichen [Desbenoit 04c, Desbenoit 04a](which often also compete for space) or by modeling autumn leaves covering vegetation or objects in the scene. [Desbenoit 04b]

Also in ecology a great amount of work has been done to simulate ecosystem development. Based on mathematical growth models resulting from such work and the FON model (Field of Neighborhood) introduced by Berger et al. [Berger 00], Alsweis and Deussen developed an innovative technique to simulate the competition between plants in large ecosystems [Alsweis 05]. Their currently state of the art work distinguished between two cases: the symmetric competition (a double-sided interaction in which the resources are equally divided between the competitors), and the asymmetric competition (a one-sided interaction; one plant gains all the resources of an overlapping area and other plants do not have the chance to gain resources) between plants. Results are shown in Figure 2.10.a.

Tree models were imported from the interactive modeling tool Xfrog. This implies how their shape is not directly influenced by the simulation process except for a factor indicating each tree’s area of influence (depicted in Figure 2.10b). Competition in this case concentrated on the domination of plants over other species and the availability of nutrition. When dominated, instances were simply removed or their growth was reduced accordingly.
Based on the method described by Lane and Prusinkiewicz [Lane 02], Alsweis and Deussen extended their work to optimize the simulation of the competition of plants for light and also to take into account below-ground competition [Alsweis 06] (see Figure 2.10c). This easily overlooked aspect of plant competition has a considerable influence on the ecosystem development.

In a second extension [Alsweis 07], new types of competition were simulated and visualized. One takes place between a healthy plant and a plant becoming sick during the simulation process (an illustration of this phenomena is shown in Figure 2.10d), and another is involved with the growth of smaller plants in the vicinity of a nursing plant.
This brings us to the final part of this chapter, where we mention a few techniques used to visualize natural objects.

2.5 Plant Visualization

As mentioned before, the contributions reported in this dissertation mainly focus on the structural realism of the plant models and put less emphasis on their visualizations. To illustrate the results of our techniques, basic rendering methods are applied. For example, stems and branches are represented by cylindrical shapes and leaves are modeled using textured polygons. Basic lighting algorithms are employed to illuminate the scenes.

The presented algorithms are designed to produce realistically shaped plant and tree skeletons. Without much effort, photorealistic visualizations [Franzke 03, Wang 05] and even non-photorealistic visualizations [Deussen 00] of these models can be achieved. An example of a possible non-photorealistic rendering style is depicted in Figure 2.11 and results from combined research by my colleagues Fabian Di Fiore, Frank Van Reeth and myself [Di Fiore 03].

In contrast, research has been presented, focussing solely on visualization natural scenes, often without actually modeling the underlying plant structure. A nice example of such a method was presented by Behrendt et al., where the amount of polygons in the scene was reduced to a much smaller number of
2.6 Concluding Remarks

well-chosen billboards representing them [Behrendt 05]. This work was based on research done by Decoret et al. on the use of imposters for accelerated rendering [Decoret 99]. A similar method was presented by Fuhrmann et al. [Fuhrmann 05].

Another approach involves the use of point-based rendering. Gilet et al. presented such a method and visualized up to 200,000 trees at real-time frame rates [Gilet 05]. Similar results were achieved by Decaudin and Neyret, but based on volumetric textures [Decaudin 04].

We refer to [Luft 05, Luft 06, Coconu 05, Coconu 06] for readers interested in a few other methods to acquire non-photorealistic plant visualizations. Various photorealistic plant visualization methods can be found in [Qin 03, Colditz 05, Dietrich 05] and [Quan 06]. A nice survey on the different computer based representations of trees and plants for rendering purposes is given by Boudon et al. [Boudon 06].

2.6 Concluding Remarks

In this chapter we presented an overview of the most important contributions made during the past few decades to the field of virtual plant modeling. A full survey would lead us too far away from the topics presented in this dissertation, hence, we focussed on an explanation of the most important biological aspects involved in environment sensing, and those methods that made up the foundations for our research.

Many attempts have been made to incorporate the environmental influences in the growth process of plants. The majority of them, however, are based on simple heuristics or techniques that mimic natural effects and do not, for instance, consider a more accurate calculation of the incident illumination.

The next two chapters describe into further detail the contributions we made to the modeling of environment sensitive plants by taking into account several illumination issues based on a simulation of the corresponding natural process and the presence of nearby objects and plants.
3.1 Introduction and Motivation

Simulation and visualization of plant competition is an important research field, not only in ecology where it helps predicting the future and the condition of ecosystems, but also in applications such as computational biology, landscape planning, and city architecture. Additionally, the methods can also be used to achieve beautiful realistic scenes that are used in the production of films, computer games and arts.

In this chapter, we present a simple and effective algorithm\footnote{This chapter is based on “A Simple But Effective Algorithm to Model the Competition of Virtual Plants for Light and Space”, [Van Haeve 03].} that allows for the simulation of plant growth in a realistic way, while taking into account environmental illumination, spatial occupancy and the nearby presence of other plants.
Our method is highly related to work done by C. Soler et al. [Soler 01, Soler 03], who presented an extension of hierarchical radiosity with clustering, to be used for simulating plant growth taking into account illumination. Their method is supposed to be very accurate, but its implementation takes considerable care and effort. In contrast, the method we present here is extremely simple while still leading to plausible results. In addition, our method also incorporates a morphogenetically based mechanism, that “discourages” plants from growing into each others space.

After presenting the different aspects of the algorithm in Section 3.2, we will demonstrate how we managed to model and visualize certain natural phenomena (see Section 3.3). In Section 3.4 a short discussion on the presented technique will be given.

3.2 Approach

The basic idea of our algorithm is to simulate light flow during all stages of plant development, by tracing virtual photons emitted by light sources in the same way as it is done in many global illumination algorithms. Furthermore, we augment the L-System, modeling plant growth with a mechanism for determining predominant illumination directions by analyzing the distribution of the direction of incidence of photons hitting plant parts. Based on photomorphogenesis principles found in literature, the same photons are used to discourage plants from growing into each others space by penalizing the contribution of photons reflected by other plants.

The algorithm starts with an L-System, representing the overall growth process to be used by the plants. Figure 3.1 shows the L-System, used to generate some of the images in this chapter. The most important aspect of this L-System is that it is “open” [Mech 96], which gives us the opportunity to make certain decisions during the rewriting process. The figure demonstrates how module $P$ can hold a parameter $p$. Each value assigned to it, ranging from 0 to 4, provides a different way to replace module $P$. This demonstrates how an interface is provided between the L-System and the software using the L-System.

In our case, each of the four values represent a direction to grow a new part at a position on the current tree model. Our algorithm computes one of these values, based on available lighting and occupancy information, in order to make plants grow towards a specific direction.
3.3 The Supported Interactions

We already mentioned in Section 2.3 how we focus on three sorts of interactions in which plants participate: light interaction, object interaction, and neighbor interaction. In the upcoming sections we will explain how our method supports these phenomena and how we implemented them based on a few well chosen heuristics.

3.3.1 Object interaction

In Prusinkiewicz 94 a basic L-System based solution can be found to mimic the interaction of a tree with a bounding volume. The approach presented checked, by means of special purpose production rules, whether an intersection between the plant and a mathematical boundary occurred. When such an intersection was detected, the new branch was cut off and a pruning signal was propagated downwards to the lower branches. Recursively, other parts were pruned and other growth directions were tested. This approach led to similar behavior as illustrated in Figure 3.2 and resulted in a more synthetic plant shape often found in topiary gardens.

However, the illustration depicted, together with Figure 3.2 are the result of our own, new method, where we used open parameters to query for object collisions with the help of an octree refined towards the objects that make up

Figure 3.1: One of the stochastic, parametric, context sensitive, open L-Systems, we used to model certain environment sensitive trees.
Figure 3.2: A synthetic plant shape based on an enclosing primitive: (left) after 4 re-writings; (middle) after 10 re-writings; (right) top-view, after 15 re-writings.

Figure 3.3: Trees interacting with objects in their surrounding environment: (left) front view; (middle) top-view; (right) front-right view.

the environment. Hence, not only a mathematical boundary can be respected easily, but even the full geometry of the scene surrounding the plant can be taken into account.

We use a similar idea to prevent the competition of several branches for the same space. To this extent, the geometry of a tree is sorted into a second octree data structure (see Figure 3.4). Each time a new branch is added, this octree is checked to see if there is any space left for a new branch to grow. If not, the new branch is cut off and a corresponding pruning signal is sent. This second octree data structure is also used to speed up ray-tree intersection testing.
3.3 The Supported Interactions

3.3.2 Light Interaction

Light interaction is one of the most important aspects of plant growth, because it provides a plant with the necessary materials to survive, by performing photosynthesis (a chemical process in the leaves requiring light). Therefore, we continue by describing a method to estimate the environment illumination.

Use of Photons. First, we need an easy to use model to represent the illumination in the scene. To simulate nature, we decided to position a light source into the scene emitting virtual photons. Each of the photons get a direction assigned and an certain amount of energy. Next, these photons are traced through the scene, taking into account how with every intersection a new direction needs to be calculated and a portion of the energy is absorbed by the object. We relate the number of photons emitted to the number of tree intersections required (i.e. keep shooting photons until you have an average of $n$ photon hits per tree part).
Photon Count. Intersections with branches receive a special treatment. Every branch in the tree provides a number of possible growth directions (for our examples we took 4 growth directions per branch), calculated from the bounding cylinder surrounding each branch segment. For some applications it might be interesting not to use a predefined amount of growth directions, and the employment of the L-Systems should be altered accordingly. Instead of one single parameter indicating the new grow direction, the angle itself could be exchanged between the software and the L-System.

Each of the growth directions of a branch receives an associated counter representing how much light arrives from that direction. Once a photon hits a tree part, the incoming light ray is compared to each of the available growth directions. By calculating the dot product of the incoming ray direction and the allowed growth direction we get an estimation of how important the contribution of the ray is to that direction and we add it up to the corresponding counter.

After evaluating all the possible growth directions with the incoming photon paths, we get a global overview per branch of the incoming light. The normalized vector corresponding to the counter with the highest value, becomes the new growth direction for that specific branch.

The most important improvement resulting from this approach is how we can model trees and plants subjected to indirect illumination, including light rays that reach them through reflections on objects. As a result, such rays can still influence trees that are blocked from a light source. An example of this phenomena is given in Figure 3.5 which depicts a plant growing around a wall to reach for more light. Initially, it started to grow towards the reflection on the wall.

3.3.3 Neighbor Interaction

Section 3.3.1 described a method to discover a posteriori how certain regions of space are overloaded with plant parts. From biological experiments it turns out that plants also anticipate to this problem by growing away from each other in an early phase [Kendrik 86]. We present a simple but effective algorithmic method to simulate this behavior.

Again, the solution is based on the light ray approach. The reason why two trees tend to grow away from each other, results from a reaction to certain rays, originating at one tree and intersecting another tree, and contain only those particular components of light that were not absorbed and thus cannot be used for photosynthesis. Those photobiological inactive parts of the light
3.3 The Supported Interactions

Figure 3.5: A plant growing in an environment with indirect illumination: (top) the plant in a random view; (bottom left) an alternative view; (bottom right) top-view with the illumination shown. Note, how the plant first grows towards the reflection on the wall and bends towards the light source afterwards.

spectrum are of no use to the leaves and will be reflected. The majority of light beams containing the useful light components will arrive from parts of the scene where a lower amount of other plants is situated.

A possible way to handle this in a plant simulation is to take into account the different wavelengths of the emitted light. In this way reflections and absorptions depend on the material properties of the scene, which demands a much more detailed description of the scene. To preserve a sufficiently simple illumination model, a different approach can be taken.

The idea is very intuitive. Whenever a light ray is emitted from a certain
Figure 3.6: Rays preventing tree parts from growing in each other’s space by exchanging information about their positions: (a.) within a single tree; (b.) between to adjacent trees.

tree part (after a reflection) a flag is set, to indicate how this ray has the potential to be a tree-to-tree ray. Every time a light ray strikes a tree part, this flag is checked to conclude whether this ray has originated on another tree part or not. In case of such a particular ray, the intersection with the second tree is handled in a different way. Figure 3.6 illustrates the rays emitted from one part of a tree and striking other parts.

Instead of increasing the counters (associated with the growth directions) we decrease the probability for that specific direction to become the new growth direction. This can be done by storing a new counter for each growth direction, indicating the penalty for growing in that direction. In the end, when the decision needs to be made about the new growth direction for each branch, the corresponding counters are added together.

To illustrate how this simple technique produces plausible results, we virtually planted 4 trees close to each other and added a light source coming from up left. Figures 3.7 and 3.8 show the resulting growth responses. Seen from the top, with the trees standing close to each other, it looks as if we are observing one single tree, because the branches have positioned themselves to an almost uniform distribution. If we take a closer look (by increasing the mutual distances between the trees, as illustrated on the right side), we can
3.3 The Supported Interactions

Figure 3.7: 4 trees competing for space: (left) the trees on the positions were we planted them, (right) the same trees but with increased mutual distances to show their competition for space.

Figure 3.8: The same tree models as in Figure 3.7 but seen from a top view. Lines are added to visualize an estimation of the space occupied by each of the individual trees.
The Competition for Light and Space

Figure 3.9: Attraction towards light and neighbor repulsion visualized on the same tree model: (left) a random view; (middle) top view, with a visualization of the attraction towards the light source; (right) top view, but visualizing the repulsion between the branches of that same tree. The black arrow indicates the incoming light direction. Different colors are used to represent the contributions of the possible growth directions to each effect.

see how each of the trees has taken into account the positions and growing behavior of the other tree parts in the scene. Lines were added to visualize an estimation of the space occupied by each of the trees.

3.3.4 Light vs. Neighbor influence

Of course, care must be taken when we add the different counters together. It depends on the species of the plants we are dealing with how much attraction there is towards the light sources, and to what degree repulsion against adjacent plants is performed. By normalizing the counters (dividing all 4 counters by the largest one, and that for both kind of counters) we can simply combine the counters proportionally to the relation between attraction and repulsion. Figure 3.9 gives a visualization of these counters.

Mathematically the weighted counter can be calculated based on a simple linear combination, as follows:

\[ C_i = \alpha \cdot \frac{A_i}{A_{\text{max}}} - \beta \cdot \frac{R_i}{R_{\text{max}}} \]  \hspace{1cm} (3.1)

where \( C_i \) is the resulting counter for growth direction \( i \), \( A_i \) is the counter indicating the attraction to light for that specific branch and \( R_i \) represents the repulsion against other tree parts. \( \alpha \) and \( \beta \) are factors indicating how to
3.4 Discussion

We proposed a simple but effective extension to L-Systems taking into account environment illumination and spatial occupancy, based on an emission of photons.

Inevitably, a number of problems still remain to be solved. First of all, we make very simple and ad-hoc assumptions about how plants react to light: how must attraction and repulsion be weighed exactly, and do plants in reality react in a linear way to incident light intensities? A solution to these questions can be found in morphogenesis literature.

Second, many aspects of the algorithm presented here can be considerably refined. In particular, we need to experiment with more advanced irradiance estimation techniques to find the predominant illumination direction, for instance in the spirit of the photon mapping method [Jensen 01]. Including the
earlier mentioned wavelength dependency of plant growth is also something that could be considered. In addition, our method should be evaluated with alternative approaches to describe plant growth, for example in the style of the work done by Lintermann and Deussen [Lintermann 99].

The algorithm presented here is merely an example of how, based on very simple algorithms, the influence of the environment can be taken into account during the growth process of plants. More extensively developed versions of these algorithms could be produced to also consider the presence of leaves and flowers during the simulation, but in the end these modifications will not influence the applicability of the technique.

The next chapter describes how we improved our method to a technique that estimates the illumination more precisely, by calculating the ray density around growth positions on the plants.
4.1 Introduction and Motivation

Light interaction is one of the most important factors in developing realistic plant models. Plants react to received illumination by bending branches, adapting their growth rate, orienting leaves and flowers, producing larger or smaller leaves, etc.

When applying the method presented in Chapter 3 of this document on a higher developed and more detailed plant model (including large amounts of branches, leaves, flowers, etc.), a very large number of rays is required to reach a sufficient average amount of rays hitting the different tree parts. Consequently the previously presented algorithm becomes quite heavy to calculate and the simplicity of the method is affected. An improvement is required to release the dependency of our algorithm on the number of rays emitted.

Hence, we present in this chapter\(^1\) a novel approach to simulate plant

\[^1\text{This chapter is based on “A Ray Density Estimation Approach to Take into Account Environmental illumination in Plant Growth Simulation”. [Van Haevre 04b] [Van Haevre 04a].}\]
growth as a response to this environment illumination.

The new algorithm we introduce calculates for each point of interest the amount of light present in a certain neighborhood, based on the estimation of the ray density and the retrieval of the mean direction of the incident illumination. Furthermore, the restriction to a fixed number of growth directions is removed allowing the plants to grow in the best direction available.

This chapter is organized as follows. Section 4.2 gives an overview of our new approach. Next, Section 4.3 clarifies how we represent the environment illumination into a well chosen data structure, and Section 4.4 concentrates on the method we used to query this data structure to retrieve information about the illumination. In Section 4.5 we demonstrate the ease by which several phototropism related phenomena are simulated with the obtained information from our new algorithm. We end this chapter with an evaluation of some results (Section 4.6) and a discussion on the presented method (Section 4.7).

4.2 Approach

Again, the underlying idea of our algorithm is to simulate light transport in the environment in which plants grow by tracing light particles originating from light sources. However, this time both the intensity and mean direction of the incident illumination are determined easily, based on a ray density estimation of the environment illumination and by means of a predominant illumination direction.

Our approach takes into account both direct and indirect illumination and is an algorithm that is both flexible and accurate. It is easy to implement and more general illumination models can be incorporated in a straightforward manner. Furthermore, using a non-uniform, adaptive data structure for storing the rays, calculation time and storage requirements are kept within reasonable limits.

The algorithm presented is inspired by the photon mapping technique [Jensen 01]. Similar to photon mapping, the algorithm will deal easily with non-diffuse light reflection and translucency as well as complex and dynamic geometry. Unlike photon mapping, we perform density estimation of light rays in an environment rather than estimating the density of photon hits on object surfaces. Moreover, contrary to other simple solutions that incorporate environment light in plant growth [Mech 96, Beneš 02b] and our earlier work
represented in Chapter 3, the new algorithm can provide accurate estimation of the illumination, including indirect illumination, with ease.

An adaptive spatial data structure is used to store the rays along which light particles travel in space. This data structure allows an efficient calculation of the ray density at locations where the algorithm needs to query incident illumination.

We use L-Systems \cite{lindenmayer1972} to model the plants and their growth movements. However, other methods can be used, as long as they support the interaction of the model with the presented data structure (e.g., marking of growth positions).

Plant models are regarded as static objects during light computation. Every growth step allows the plants to change their shape according to the lighting in their neighborhood. The changes in the plants’ shapes influence the lighting in the scene which again influences the plants’ shapes etc. In order to speed up this iterative process, we create a data structure that facilitates the following two actions:

1. Avoiding full re-computation of the lighting after applying the changes to the plants’ shape;
2. Fast querying of the lighting in the neighborhood of a plant, needed in order to determine how the shape of the plant is going to evolve.

\section*{4.3 Representing the Environment Illumination}

Light transport is simulated by tracing light particles (i.e. photons) originating from light sources. As these photons strike a surface they can either be absorbed or reflected back into the scene, according to the material properties of the surface. In the next section, we propose density estimation of the rays (along which particles travel) in order to calculate the intensity and mean direction of illumination in space (i.e. the radiance field).

For a start, a data structure allowing efficient density querying needs to be set up. This data structure should allow for the rapid retrieval of rays passing nearest to a query location. Spatial subdivision data structures such as a uniform grid, an octree or a BSP tree, in which spatial cells are tagged with references to the rays passing through them, are well suited for this task.

The use of a uniform grid is prohibitively expensive in terms of memory requirements and inefficient as well, since many light rays are likely to pass through areas with no active plant growth, and in which no querying for
Figure 4.1: An octree as a spatial subdivision data structure, allowing local storage of references to nearby light rays.

illumination will be performed. Each ray would intersect a lot of grid cells and, hence, each of the grid cells would have to keep a reference to the ray. Therefore, the use of an octree is recommended, since it is a non-uniform data structure that can be updated dynamically (see Section 4.3.1) and can be constructed with higher detail where required (see Figure 4.1).

The octree is constructed in a demand-driven way. Whenever the radiance field needs to be queried at a given location, a “point of interest” or “refinement point” \( p \) is constructed. Every time such a new point is added to the scene, the octree is refined (if necessary) by splitting cells into uniform octants, until a specific cell width \( c \) is reached.

When light is emitted from the light sources, light rays intersect the existing nodes of the octree. After adding a refinement point and updating the references to the rays, the cell containing the point will hold a reference to all rays passing through this cell. Whenever a refinement point is no longer required, the references to the intersecting rays can be discarded, while keeping the references in the higher layer of the octree for further refinements.

Consequently, octree cells might be created as refinement points are added.
4.3 Representing the Environment Illumination

After their addition, the cells are updated using the references to rays stored in higher levels of the octree. When light rays are added to the scene, the topology of the structure remains unchanged and only one update of the existing cells is required. Empty cells (which contain no detail points) are deleted recursively. This way, the amount of memory required is kept to a minimum.

4.3.1 A Dynamic Process

While the plant’s shape is enlarging, several branches, leaves and flowers are added to the scene. As a result, a small number of rays suddenly gets occluded, leaving some invalid paths in the environment. To resolve this instability, a dynamic update of the rays is required.

Refinement points are added to positions where the plant is growing. Consequently, the topology of the grid is focussed around these growth positions, and the rays that might be blocked by new plant parts are found in the corresponding grid cells of these points. As a path consists of a linked list of rays, it can be broken off as soon as an interruption occurs. Hence, the pruned part of the path is discarded, references to these parts are removed from the octree, and the path is extended by recasting the intersected ray into the scene, taking into account the new plant parts.

Listing 4.1 illustrates the steps required to update a plant model to the next stadium of its growth, after being initialized:

1. Update the sequence that represents the shape of the plant, using the production rules of a plant’s specific L-System that describes the overall behavior of the plant.

2. Create a plant model from this sequence and add refinement points at query positions (indicates by communication terms in the symbol string). As new cells are created, the shape of the octree and the references it contains are updated.

3. Remove empty cells from the octree whereas some of them might exist from previous iterations but have become useless for the current plant shape. Hence, unused memory can be released.

4. Remove all ray references in the finest layer of the octree (and the remainder of the paths constructed behind them) because these are the locations where new plant parts may have been added or altered.

5. Due to these removals, paths that were cut need to be extended again, incorporating the geometry of the new plant shape.
Plant Growth based on a Ray Density Estimation

```c
Grow(Plant) {
  LSystem.LoadFromFile();
  LSystemSequence = LSystem.getAxiom();
  for each growth iteration do
    LSystemSequence.update(LSystem);
    // using a plant’s specific L-System
    Plant.CreateModel(LSystemSequence);
    // while adding refinement points at query locations
    Octree.RemoveEmptyCells(); (recursively)
    // to free memory
    Octree.RemoveRayReferencesInQueriedCells();
    // top-down to remove all references to obsolete rays
    Rays.update();
    // recast paths where broken
    LSystemSequence.RetrieveOpenParameters()
    // updates the sequence while calculating ray density
    // and incident light direction
    Octree.RemoveRefinementPoints();
    // allowing obsolete cells to be removed in the next iteration
  end for
}
```

Listing 4.1: Overview of the method we used to process the growth of an environment sensitive plant, illustrating the procedure to keep track of the status of the illumination in the scene.

6. Using the constructed octree, illumination intensity and mean incident light direction can be calculated (Section 4.4) and returned to the L-System by means of “open” parameters.

7. Remove all refinement points as they are no longer required during the next iterations.

4.4 Querying the Environment

In this section, we explain how to retrieve the light intensity and mean incident light direction while estimating ray density.

Each ray cast in the scene transports a fraction $\Delta \Phi_i$ of the light present in the scene. Based on a single ray, we cannot trace the dominant direction and intensity of the light flow at a specific point. To calculate this direction and to find an estimation of the light intensity, the ray density is used.
method is inspired by the estimation of the irradiance on a surface, described by Jensen [Jensen 01].

The ray density at a position can be expressed as the amount of rays intersecting a predefined volume (a sphere in our case) around that point. To estimate the density of rays around a point, several possibilities exist [Silverman 86], for instance:

**The Nearest Neighbors Method:** search for a specific number of the closest rays around the point and calculate the corresponding volume. The radius equals the highest distance found between the point and one of these rays.

**The Histogram Method:** search all rays around the point inside a fixed volume. This means that a fixed search distance is used.

In both cases, the illumination intensity at the query location can be obtained by summing the photon powers $\Delta \Phi_i$ of the rays, per unit of projected query volume surface area. When using a sphere as a query volume, the projected surface area is just the area of a disc with the same radius. When the Russian Roulette reflection algorithm [Jensen 01] is used while tracing the light paths through the environment, the power $\Delta \Phi_i$ of all the rays can be kept equal and the ray density can be expressed as the amount of rays per unit of projected query volume surface area. This estimation can be used relative to a reference density (representing full illumination). In this way one can decide how much illumination is present at a certain point. The reference density can be acquired at a user-defined position and should represent the amount of rays per unit of projected query volume surface area for a fully illuminated point.

The dominant illumination direction is found by summing vectors with magnitude $\Delta \Phi_i$ (possibly equal for all rays, when Russian Roulette is used) into the ray direction. The result should be normalized.

In order to facilitate the search for rays in the octree we advise employing the histogram method, since the fixed search distance allows a much easier search for rays in the octree. Using a fixed search distance $d$ to estimate the ray density, we can calculate a depth for the octree to obtain a cell width of maximum $2d$ for the lowest layer (the layer containing the refinement points). Consequently, we can restrict our search for rays to the current cell and its 26 adjacent cells (see Figure 4.2 which for clarity visualizes the 2D case).

Some of the required adjacent octree cells might not yet exist. To force their existence, (as we need the references to their intersecting rays), we add
Figure 4.2: An adaptive octree depth based on a user-specified search distance $d$, used for illumination querying: (left) $2d$ is smaller than $c_1$, the octree needs to be divided; (right) the updated situation: $2d$ is larger than $c_2$, no further subdivisions are needed.

The necessary refinement points to the octree. The positions can be calculated from the query position. As a result, we locally create a uniform grid which allows us to find the requested light rays by simple look-up. Of course, there is some cost involved in refining the octree at newly inserted refinement points, but often subsequent queries are done in the same neighborhood, usually spreading the cost over several queries.

From the rays intersecting the 26 octree cells, only those intersecting the search volume (a sphere) are retained. This requires checking the distance between the rays and the query position, and comparing this distance with the search sphere radius. Figure 4.3 shows how an octree in combination with a local uniform grid allows a fast, position-oriented search in space for light rays.

In the next section we show how many interesting illumination related plant growth effects can be modeled considering little more than the intensity and mean direction of illumination near active locations of growth. More complex interactions can be simulated as well, but require more knowledge about plant specific growth behavior.

4.5 Examples of Supported Phenomena

The examples presented in this section are not based on specific properties of certain species, but illustrate common aspects of plant growth concerning light
Figure 4.3: Three incremental refinements of the nearest rays around a query point: (a.) all rays within the scene; (b.) all rays intersecting the 27 surrounding grid cells, (c.) all nearest rays within search distance $d$. 
interaction. They demonstrate how the combination of existing literature on L-Systems [Lindenmayer 72, Prusinkiewicz 97] together with research on the biological properties of plants [Kendrik 86] allows for the creation of growing plants in a natural way within their environment.

In this chapter, we focus on responses corresponding to phototropisms, which are phenomena manifested as a sequence of growth movements by plants or a change in their shape in relation (positive or negative) to the incoming light direction.

Several movements are made by a plant while reaching to or moving away from the available illumination. Branches elongate or remain short due to the amount of light present in their environment. Many existing plant species bend towards light sources, to reach for as much of the available light as possible. Others don’t or do the opposite. Leaves might position themselves perpendicular to the incoming light direction while gathering energy for the growth of the plant – by means of the chemical process called photosynthesis. Other species, however, orient their leaves away from that direction to prevent overheating, and allow light beams to penetrate more deeply into the canopy. Also, the size of the leaves may correspond to the availability of light at a specific position.

4.5.1 Branch Bending

The bending of branches due to the incoming light direction is a well known phenomenon. Several plants try to shape themselves to increase the number of plant parts that can be reached by light rays. In order to enlarge the total surface of leaves and branches that are illuminated, the branches will bend to occupy free space providing sufficient illumination to the rest of the model.

When using L-Systems to model the plants, communication symbols can be used to interact with the environment [Mech 96]. For instance, if $F$ represents a branch, an “open” symbol $R(? , ?)$ can be placed behind it. $R(? , ?)$ will be interpreted by the application as a request for a new growth direction. The parameters are given a proper value resulting is $R(r, p)$. Using the L-System from Listing 4.2 the symbol will be replaced by a combination of rotations $\langle r\rangle \& \langle p \rangle / (r)$, due to production rule $p_2$. As a result, the state at the end of the branch will have been turned to the calculated direction.

The parameters $r$ (roll) and $p$ (pitch) are calculated as follows: whenever the symbol $R(? , ?)$ is encountered while interpreting the sequence, a refinement point is added to the octree at the current state’s position. To find the values
4.5 Examples of Supported Phenomena

\[ \omega : A \]
\[ p_1 : A \rightarrow FR(? , ?)A \]
\[ p_2 : R(r, p) \rightarrow \backslash(r)\&(p)/(r) \]

Listing 4.2: L-System 1, illustrating how branch bending is achieved.

for the open parameters in the new sequence, the nearest rays to this position are located.

From these rays the mean incident light direction can be derived and compared to the current state vectors \( \vec{H}_c, \vec{L}_c, \) and \( \vec{U}_c. \) A roll angle (rotation around \( \vec{H}_c \)) and pitch angle (rotation around \( \vec{L}_c \)) are calculated, which transform the current state to a new state that incorporates the requested bending. To suppress the effect of bending too exaggerated, a maximum pitch angle \( m \) can be defined and provided to the application instead of one of the question marks (e.g., \( R(? , m) \)), to which the calculated pitch angle can be constrained.

The combination of the \( \backslash(r) \), \( \&(p) \) and \( /(r) \) commands makes it possible to rotate \( p \) degrees away from the heading vector, in any direction, without a distortion of the overall plant topology.

The set of sequences displayed in Listing 4.3 is an example of what might be produced during a few derivation steps (we limited the pitch angle to 8 degrees):

\[ 0 : A \]
\[ 0' : A \]
\[ 1 : FR(? , 8)A \]
\[ 1' : FR(20 , 6)A \]
\[ 2 : \backslash(20)\&(6)/(20)FR(? , 8)A \]
\[ 2' : \backslash(20)\&(6)/(20)FR(9 , 5)A \]
\[ 3 : FR(20)\&(6)/(20)F\backslash(9)\&(5)/(9)FR(? , 8)A \]
\[ 3' : FR(20)\&(6)/(20)F\backslash(9)\&(5)/(9)FR(3 , 8)A \]

Listing 4.3: A possible result from the processing of L-System 1. Each line \( x \) represents one rewriting step. The lines \( x' \) following them are their updated versions, containing the calculated and constrained rotation values derived from the queried illumination information.

When visualizing such a sequence using turtle graphics, a growing branch is displayed bending locally towards the direction calculated by the application.
Figure 4.4: The schematic representation of a branch bending towards the dominant illumination direction, based on a piecewise calculated roll and pitch angle.

This phenomenon is illustrated schematically in Figure 4.4.

4.5.2 Leaf and Flower Orientation

As mentioned before, the leaves are one of the most important sources of energy for the plant. When they absorb light, the process of photosynthesis produces energy which makes it possible for the plant to grow, to remain strong, to produce new plant parts, etc. In order to sustain this task, an optimal orientation of the leaves in relation to the positions of the light sources becomes very important.

When a leaf is added to the end of a branch or a twig, a refinement point is created and placed at the corresponding position into the octree. Next, the nearest rays are located and the mean incident light direction is estimated. In
response, a new state is created for this position, based on the current state of
the L-System and the estimated incoming light direction. Finally, the changes
required to convert the original state to the desired state are returned to the
L-System to apply them in the next iteration.

The same idea is valid when positioning flowers or other plant parts.

4.5.3 Branch, Leaf and Flower Size

Another aspect of phototropism we want to illustrate with our algorithm,
concerns the length of the branches and the size of the leaves. Several types
of plants tend to grow faster (enlarging the branches) in dark spaces while
preventing leaves from growing, in order to reach the illuminated areas faster.
Once they escape from a darker region, they reduce their growth speed and
focus on the production of leaves and flowers, to obtain their required energy.

Again, we used a special symbol in our L-System which allows the required
interaction with the environment. L-System 2 from Listing 4.4 contains the
main production rules for this behavior (where \( f_i \), \( g_i \) and \( h_i \) are plant related
functions in \( i \)).

\[
\omega : AI(?)F(0)I(?)W(0)\\np_1 : A \rightarrow I(?)F(0)[+(8)I(?)L(0)]A\\np_2 : I(i) < W(0) \rightarrow W(f_i)\\np_3 : I(i) < L(0) \rightarrow L(g_i)\\np_4 : I(i) < F(0) \rightarrow F(h_i)
\]

Listing 4.4: L-System 2, consisting of production rules capable of interacting
with the environment, by means of a left context.

In the axiom, the open symbol \( I(?) \) appears twice. First before an \( F \) (a
piece of a branch) and next before a \( W \) symbol, which marks the position of
a flower in the plant. In the first production rule it also appears before an \( L \)
which in our case is the symbol representing a leaf.

Whenever \( I(?) \) is encountered, a refinement point is added at the position
indicated by the current state of the L-System. The nearest rays are located
and this time, only the ray density is estimated. Depending on this density a
value is returned for the open parameter. Subordinate to this parameter, spe-
cific changes can be made to the plant model. Rules \( p_2 \), \( p_3 \), and \( p_4 \) demonstrate
how the parameters of the module representing a flower, a leaf or a branch can
be altered according to the value \( i \) that is returned by the application through
module $I$. A plant specific function $f, g$ or $h$ can map this parameter on a suitable value.

When using this simple technique, the adaptation of a plant to darkness and its behavior to reach for more (or less) illuminated environments by increasing or decreasing the size of its components, can be simulated.

4.5.4 Complex Behavior

All of the previously mentioned changes to the plant model can easily be combined into a more complex behavior. It simply requires to provide an L-System containing the production rules which describe the needed behavior. For instance, by combining the two techniques described earlier in this chapter (Sections 4.5.1 and 4.5.3) into one L-System, branches can bend or remain straight depending on both the actual amount of illumination reaching them and the light’s mean direction. The combination of production rules illustrated in L-System 3 (Listing 4.5) ensures the maximum bending angle for module $R$ to be limited to a function $f$ of parameter $i$, which was returned by the query for ray density proceeding it.

$$
\omega : A
p_1 : A \rightarrow I(?)WF(1)A
p_2 : I(i) < W \rightarrow R (?, f_i)
$$

Listing 4.5: L-System 3 combines the techniques used in L-Systems 1 and 2.

This allows a plant to bend relative to the amount of incident illumination. It is obvious that other phenomena can be simulated easily using this kind of combinations (e.g., linking the maximum bend angle to the age of the branch, increasing or decreasing the amount of leaves or flowers in relation to the available light at a certain position, ...).

4.6 Results

Figures 4.5 and 4.6 show two test scenes. All plant models in one scene are created using the same L-System, describing the plant specific topology and growth behavior. Due to light interaction, the shape of each plant is altered, according to its position in the scene and the location of the light sources.

The corresponding charts indicate the amount of time required for each iteration in the plant growth simulation. They display the calculation speed
Figure 4.5: (a.) Several plants, created from the same L-System, growing in various ways due to different environmental properties: under a table, on a table and closer to the light sources, next to a wall, etc. (b.) The calculation timings required to update the model.
Figure 4.6: (a.) A similar result as displayed in Figure 4.5 but based on a different L-System. (b.) The corresponding chart depicting the calculation timings.
4.6 Results

of a full update of the plant model (including the update of the octree, the refinement points and the rays). We must stress that these results can be improved using a more optimized implementation.

The shape of the curves clearly imply that the first iteration of the growth process relatively demands more calculation time than the subsequent iterations. This is due to the first refinement of the octree, which demands the creation of several octants at different layers and a recursive update of the references to the intersecting rays. The other iterations have a calculation time increasing exponentially in relation to the complexity of the model. The small fluctuations in the illustrated timings are caused by larger changes to the octree which are necessary when the model enters previously non existent parts of the octree. Again, the corresponding refinement requires the creation of several octants at different layers and an update of the corresponding references to the intersecting rays.

The first scene, depicted in Figure 4.5 contains four plants with flowers, subject to the following rules as they grow:

- Branches bend towards the mean incident light direction;
- The maximum bending angle depends on the amount of illumination at each position (when more light reaches the branch, less bending occurs);
- Flowers are created when a sufficient amount of light is available.

Three small bushes were added to the scene from Figure 4.6 growing according to the following behavior:

- Branches bend towards the mean incident light direction;
- The size of branches and leaves depends on the amount of incident illumination.

Figures 4.7 and 4.8 provide a closer view on the resulting plant models. The first set illustrates how a simple plant model with small flowers is influenced by its environment. Illustration ‘a’ depicts a plant positioned on the ground, close to a wall. As a result, the plant model bends away from the wall, and reaches towards the open space in the scene where it can access more of the available illumination. The result displayed in image ‘b’ demonstrates how an instance of the same plant species, when positioned in the center of the scene, nicely spreads out his branches and leaves into the available space, because
Figure 4.7: Different models generated by applying our ray density estimation technique: (a.) a plant model bending away from a wall, to access more of the available illumination; (b.) the same species, spreading out its branches and leaves, as the illumination is equal from each incident direction; (c.) the reaction of this plant when it is positioned underneath a table, blocked from the light sources.
Figure 4.8: Another set of plants generated using our ray density estimation technique: (a. and c.) A bush positioned on top of a table, next to a wall. While growing, the plant model produces a denser canopy on the higher illuminated side and remains less evolved on the side closer to the wall; (b.) the bush positioned into the center of the room, allowing for a more equal development in all directions.
Figure 4.9: Several snapshots of an animation depicting the growth of the plant under the table from Figure 4.7.c. Both the maximum bending angle and the creation of new flowers depend on the amount of incident illumination at each specific position.

the illumination is equal from each incident direction. Image ‘c’ depicts the reaction of this plant species when it is positioned underneath a table, blocked from the light sources. First, the plant grows away from its occluded position, by bending its branches towards the side that provides a sufficient amount of light to survive (by means of indirect illumination). As the model enters higher illuminated areas in the scene, elongation of the stems is reduced and the plant evolves further by producing more leaves and flowers.

The second set of images, depicted in Figure 4.8 represents similar behavior, but achieved on a small bush model. This time, image ‘c’ demonstrates how a bush seed was planted on top of a table, next to a wall. Consequently, while growing, the plant model produces a denser canopy on the higher illuminated side and remains less evolved on the side closer to the wall. Image ‘a’ shows a top view of this situation. Similar to image ‘b’ from Figure 4.7, image ‘b’ displays the bush positioned into the center of the room, allowing
4.7 Discussion

for a more uniform development in all directions.

Figure 4.9 displays a few snapshots of an animation depicting the growth of a plant located under a table, similar to the example from Figure 4.7c. As the amount of incident illumination increases, the bending and elongation of the branches is reduced and the plant develops further by creating more flowers and larger leaves.

4.7 Discussion

In this chapter we presented a novel approach, based on a ray density calculation, to acquire an estimation of the environment illumination and the predominant illumination direction. This information is employed in the simulation of plant growth, allowing several responses of plants to their environment while obtaining an optimal growth. The flexibility and accuracy of the algorithm, together with its low calculation time and limited memory usage ensure a useful technique, attractive for plant modeling applications.

Our ray density estimation exhibits noise, due to randomness in the light emission. This produces fluctuations in the growth parameters which are hardly visible and in some cases even provide a small increase to the realism of the grown plants. We believe this noise can be reduced by using a different estimator. The nearest neighbors method, for instance, always guarantees a sufficient number of rays to do statistics.

Furthermore, exploiting data available from real plant species could remove several of the assumptions we made about the biological reactions of plants to environmental influences.
Part II

Plant Animation
Video Textures of Plants

As a first topic on the representation of moving natural objects, we introduce a novel algorithm to create video textures of plants, specifically designed to easily represent their dynamics in a realistic way.

5.1 Introduction and Motivation

In the year 2000, the first method was described to generate Video Textures [Schödl 00], which are short perpetual video sequences, often repeatable in time and synthesized from existing video data.

In the original method described by Schödl et al., all frames of a given video sequence were compared pairwise, based on a distance metric calculated on the image data. Sufficiently matching frames acted as transition points where the video playback could deviate from its original path by means of a rearrangement of the video data. However, this method is only applicable when a sufficient number of smooth transitions between distant frames can be made. When not available, the creation of a video texture with Schödl’s method becomes impossible.
Even though other methods for video texture creation have been proposed during the following years, the applicability of all these techniques is still restricted to a small set of possible inputs. Footage of plants, for example, take no part in this history, due to their often turbulent motions and the resulting dissimilarities between the different captured frames.

5.2 Our Contributions

In response, this chapter presents an alternative technique for the creation of video textures, focussing on plant animation. By exploiting the symmetric properties of the frames in a given video sequence, instead of their similarity to other frames, a new perpetual video stream is generated by reversing video playback when appropriate. Hence, we formulate our main contributions to the video-based animation of plants as follows:

- An extension to the set of possible input videos for video texture creation;
- No explicit requirement for actual transitions in the video data, making it suitable for several natural phenomena;
- The applicability of the technique as an extension to the original method by Schödl et al., allowing for a hybrid technique that further increases the randomness and, hence, improves the realism of the achieved results;
- Less visual artifacts caused by varying lighting conditions during video capturing;
- An adaptive algorithm for an optimal use of the available frame space;

The remainder of this chapter is organized as follows. Section 5.3 elaborates on work related to ours and Section 5.4 reveals the underlying concept of our technique. Next, after a short overview of our approach in Section 5.5, Section 5.6 explains how we performed the analysis part of this method. Section 5.7 demonstrates how video textures can be extracted from the obtained information and Section 5.8 describes some examples. We end by discussing the proposed technique in Section 5.9.

\footnote{This chapter is based on “Video Textures Exploiting Symmetric Movements”, \cite{VanHaever05b}}
5.3 Other Work on Video Texture Creation

After presenting their first work on the creation of video textures, Schödl et al. extended their own method to the use of video sprites within constrained animations and machine learning for video based rendering, resulting in more interactive animations [Schödl 01, Schödl 02]. In [Kwatra 03], video texture synthesis was performed based on a graph cut technique.

A different method was presented by Campbell [Campbell 02], who synthesized video textures using PCA dimensionality reduction and applied an auto-regressive process to move through the lower dimensional manifold to which the videoframes were transformed.

Similar to Campbell’s Method, the applicability of video textures was extended by de Juan and Bodenheimer towards cartoon textures [de Juan 04]. Borrowing ideas from video textures they also used a distance metric between pairs of existing cartoon frames, to rearrange a given set of data. To acquire a sufficient number of smooth rearrangements of cartoon data, a large amount of input data was required and mapped to a lower dimensional manifold. Within this image space, the shortest path was extracted between any two frames.

However, several issues could be identified. First, a very large data set of input frames (sometimes more than 1000 frames) was required to ensure a decent variation of the animation. Next, a large amount of preprocessing was needed to reposition the image data and apply background removal. Moreover, the generated cartoon animations also demanded a postprocessing step, in which gaps (due to insufficient cartoon data) needed to be filled with some extra user input, and visual discontinuities were removed.

In 2006, their followup work [de Juan 06] improved the segmentation procedure for background removal and extended the given set of traditional animation data with automatically contour-based in-betweening data. Hence, the required postprocessing by the animator was reduced significantly.

One year earlier, Agarwala et al. introduced Panoramic Video Textures [Agarwala 05]. They described a mostly automatic method that took the output of a single panning video camera and created a panoramic video texture: a video that has been stitched into a single, wide field of view and that appears to play continuously and indefinitely.

Our research aims for a larger set of possible input videos for video texture creation, including plant movements containing no smooth transitions, by exploiting the symmetric character of their dynamics.
5.4 Video Textures of Symmetric Motions

An important property of a video texture is the possibility to repeat the generated sequence over time without noticeable, disturbing repetitions of the same movements. This requires the displayed motions to be made of actions that remain natural looking when repeated continuously. Such movements are often focused around one or more fixed positions to which the object returns from time to time.

Standard video textures exploit these reference locations by means of transitions [Schödl 00]. Whenever a similar position of an object or set of different objects is detected on different video frames, a transition can be made, resulting in a rearrangement of the frames of the original video sequence.

In our case, bearing in mind our goal to achieve realistic plant motions, the observation of the movements of many natural objects in our environment motivates the exploitation of their symmetric character for video synthesis.

In a symmetric context, the movement away from a reference position and the returning movement are (sometimes exact) opposites. Which is not always true for the complete recorded motion, but most of the time it holds for several frames in the neighborhood of the point where the motion turns. This implies how, while playing the video sequence, a large number of frames can be reused when an extreme position in the captured motion is reached, by reversing the playback of that action. This property is illustrated in Figure 5.1 on a candle flame example.

Hence, the use of video textures can be extended to motions containing no smooth traditional transitions. Motions of plants form a typical subset of this
5.5 Approach

It is very unlikely to find exactly the same configuration of plant structures on different locations in a video, due to their complex branching structure, turbulent motions and the incoherency of the movements of the individual plant parts. On the other hand, plants tend to move very symmetrically. Their supporting structure forces them to return to their original state whenever an external force is applied to them, by means of successive decreasing symmetric movements, similar to the motion of a pendulum.

5.5 Approach

The method described here exploits these symmetric properties and can be divided into three steps:

1. Locate highly symmetric frames within the video data (Sections 5.6.1 and 5.6.2);
2. Derive statistics from this data, indicating the probability for each frame to reverse video playback (Section 5.6.3);
3. Apply these probabilities to generate a new videostream from the original data (Section 5.7).

5.6 Video Analysis

As mentioned before, we are interested in finding the specific positions in a video sequence where playback can be reversed, without noticeable visual inconsistencies. With the traditional video textures method, good transitions were found when the succeeding frames before and after the transition matched almost completely.

5.6.1 Locating Highly Symmetric Frame Positions

In the case of symmetric movements, a similar procedure can be applied. To locate highly symmetric frame locations, we compare the frames located over an equal distance before and after the test position. If these neighboring frames match sufficiently, according to a user-specified threshold, a symmetric position is marked. To grade the equivalence of the corresponding frames, we use the $L_2$-norm. Other metrics can be used as well.

The symmetric property of a location within a video sequence can sometimes be very local. Just a few frames should be included in the test. Also,
the influence of nearby frames is often more important than the similarity of more distant frames. To accomplish this, a weighted sum of the $L_2$ distances is used (Equation 5.1), by means of binomial coefficients.

$$Sym_i = \sum_{r=1}^{n} \left( \frac{2(n-1) + s}{n-r} \right) \cdot D_{i-r,i+r} \quad \text{with } s \in \mathbb{N} \quad (5.1)$$

In this equation $n$ equals the amount of frames included in the test, $s$ influences the smoothness of the binomial coefficients and $D_{i-r,i+r}$ is the $L_2$ distance between the frames located $r$ positions before and after testframe $i$. Higher $s$ values relate to higher weights for nearby frames and lesser influence for more distant frames.

Currently, a high $Sym$-value corresponds to less symmetry, resulting from larger differences between equidistant surrounding frames. To make these values more meaningful and easier to handle, we map them onto the unit interval $[0..1]$ using Equation 5.2. In Figure 5.2 we depicted these transformed $Sym$-values in gray for a simple clock example from which several stills are depicted in Figure 5.3. A value of 0 now indicates the “absence” of symmetry, whereas values closer to 1 correspond to frame locations with higher local symmetry. $Sym_{max}$ contains the highest $Sym$-value computed during the previous calculations.

$$Sym_i = 1 - \frac{Sym_i}{Sym_{max}} \quad (5.2)$$

### 5.6.2 Reducing the Number of Candidates

At this stage, we estimated for each frame in a videosequence how well it mirrors the motions contained in the frames before and after its position. Next, we need to limit the available set of frames with considerable symmetry values, according to two rules:

1. Only the frames with the highest $Sym$-values are retained;

2. The symmetric points should be positioned far enough from each other, to prevent rapid repetitions of the same short action, during video synthesis.

To achieve these goals, we only keep the subset of the original frame positions containing a $Sym$-value which is a local maximum within a user specified range $r$, according to Equation 5.3. The other values are reduced to 0.
Figure 5.2: A graphical representation of a simple clock sequence consisting of 63 frames (from which several stills are depicted in Figure 5.3). (Gray) Scaled $Sym$-values, indicating the symmetric property of each frame; (Red) Maximum derived probabilities, representing the chance for a frame to act as a turn position; (Green) Initial probabilities, used for video texture synthesis.

Frame positions closest to the beginning and the end of the video, with a sufficiently large $Sym$-value get a new value equal to 1 to indicate that video playback should always be reversed at these positions, preventing dead ends.

$$Sym_i = \begin{cases} Sym_i & \text{if } Sym_i = \max (Sym_{i-r}, ..., Sym_{i+r}) \\ 0 & \text{otherwise} \end{cases} \quad (5.3)$$

5.6.3 Extracting Probabilities

The calculated $Sym$-values can be transformed into probabilities, indicating the possibility for a frame position to act as a turnpoint. A fully symmetric frame ($Sym$-value of 1) should have a 50% chance to reverse video playback and an equal probability to proceed in the current play direction. For all frame positions we state:

$$P_{max_i} = \frac{Sym_i}{2} \quad (5.4)$$
Equation 5.4 indicates how a lower Sym-value corresponds to a smaller probability to reverse playback and a higher probability to continue. The outmost turn positions acquire the maximum probability:

\[ P_{\text{max}_a} = P_{\text{max}_z} = 1. \] (5.5)

For the clock example, we depicted these stochastic values in a red color in Figure 5.2 (which for the outmost turn points is blocked buy the green curve).

## 5.7 Video Synthesis

A straightforward algorithm to synthesize new video from the original sequence is simply to start video playback at a random position and use the precalculated probabilities to decide if a reverse is made at a certain frame position. The problem with this approach, is that most of the time the same turns will be taken, that is at the positions with the highest probabilities. Also, the same part of the original video sequence will be reused several times, leaving a portion of the available frames unused.

### 5.7.1 An Adaptive Approach

To exploit the full range of available frames, we start our video synthesis with initial turn probabilities equal to 50% of the earlier calculated maximum allowed probabilities (Equation 5.6). This increases the chance to continue with the original video playback at the turn positions, making it possible to travel further within the frame space. The values of the outmost turns always remain unchanged: \( P_a = P_z = 1. \)

\[ P_i = \frac{P_{\text{max}_i}}{2} \] (5.6)

We visualized these start values for the clock example in Figure 5.2 with a green color.

We encourage a changing behavior of the turnpoints, by increasing or decreasing the probability of a point every time it is reached. When a turn is taken, the probability to make another turn at this point is lowered with a fixed or relative amount, increasing the possibility to continue the next time without taking the turn. When a turn is not taken, the opposite behavior is provoked, by increasing the probability. These modifications happen, according to Equation 5.7:

\[ P_i = \begin{cases} \max (P_i - \text{inc}, 0) & \text{if turned at frame } i \\ \min (P_i + \text{inc}, P_{\text{max}_i}) & \text{otherwise} \end{cases} \] (5.7)
5.8 Results

The resulting rearrangements of the video frames can be performed in real-time and result in completely random video streams.

The first example, displayed in Figure 5.3, shows several stills from the original video sequence of a clock with a swinging pendulum. The same input video was used by Schödl to acquire an example for his original method. The forward and reversed backward swinging motions are almost identical, resulting in typical high symmetric turn points (see Figure 5.2). The result after applying our algorithm demonstrates how the technique manages to produce a new sequence with a quality that minimally matches the results of the traditional method. The synthesized motions remain very realistic and the video contains no visual lighting artifacts.

For the next examples only 1 still per subject is depicted. Due to the limited and sometimes turbulent motions, the effectiveness and with that the essence of our method cannot easily be demonstrated from a set of subsequent snapshots, and are best viewed as actual video sequences.

The first 3 examples (see Figures 5.4.a, b and c) are generated from video sequences of small plants and hedges, subjected to wind. Even though the movements of the individual plant parts are not isolated and the overall motion is turbulent, the video synthesis reveals no disturbing or unrealistic motions. Due to the resulting absence of good transitions, Schödl’s method fails to create a satisfying result.

The observation of a candle flame (see Figure 5.4.d) reveals another typical example of symmetric movements. This is a very good candidate for a hybrid method, in which the use of transitions is mixed with occasional turns of the video playback to increase randomness in the resulting video. Using only our new method already provides a very satisfying result, with no unrealistic motions or disturbing visual drawbacks.

The final example demonstrates how a lot of symmetric motions can be observed in our environment. A flower, for example, is moved continuously by the wind. Meanwhile, insects may be crawling on it by means of small successive (reversible) displacements (see Figures 5.4.e and f). Both movements have a high symmetric appearance. Even with the combination of both motions, the resulting synthesized video sequence shows no immediate unnatural behavior.
Figure 5.3: Stills from the original video sequence of a swinging pendulum. (a. to e.) from the outmost left position of the pendulum towards the outmost right position; (e. to i.) the reversed motion. The specular reflection on the pendulum clearly reveals the similarity between frames located at equal distances from the outmost right position, which is depicted in e. (e.g., d = f, c = g, etc.)
5.9 Discussion

The presented retrieval of the symmetric properties of the video frames and the algorithm for video synthesis are both limited to an $O(n)$ complexity, resulting in a very short calculation time. The examples described were processed within a few seconds, on commodity hardware.

The results from the original video texture method [Schödl 00] suffered from small visual inconsistencies due to slightly changed environmental conditions before and after the transitions, in terms of lighting. Because no transitions are taken with the new algorithm, and any two frames in the synthesized

Figure 5.4: Examples of possible subjects to which our method can be applied. (a. to c.) 2 plants and a hedge, moved by a wind force; (d.) a turbulent candle flame; (e. and f.) combined motions: insects crawling on a flower moved by the wind.

Because the influence of subsequent small camera movements is negligible on the overall symmetric properties of the video content, this final example also demonstrates the applicability of our method on input data captured without a fixed camera position.
Video textures are only applicable under certain conditions. The captured actions must contain repetitive motions and be isolated from other actions in the background. We extend this range of possible inputs to a larger set, including symmetric motions without actual repetitive actions, but still require the motion to be significantly isolated from other moving parts.

When applied to isolated motions of smaller plants, our technique performs very well. Even the combination of several plants can result in plausible results. A moving tree, however, can be seen as a large set of individual branches moving independently in the wind. As a result, almost no occurrences would be found in which the tree configuration is similar at different time stamps, making the rearrangement of frames very hard. Clearly, tree motions require a different approach.
The appealing quality of the resulting video textures of plants generated with our new method, together with the simplicity of the technique to create them, motivated the search for other domains where we could apply the underlying concepts of video texture creation. Hence, we decided to contribute to the creation of cartoon textures, which are the 2D animation variants of video textures.

6.1 Introduction and Motivation

Traditionally, 2D animation production has been a labor-intensive process of building up animated sequences by hand. Most work and hence time is spent on drawing, inking and coloring the individual animated objects for each of the frames.

This time-consuming work can be reduced by replacing parts of the animated scene with short repetitive animations reusing a given set of frames – a concept similar to the use of video textures.
Therefore, we present in this chapter an improved method to create perpetual cartoon animations from a small set of given keyframes, inspired by concepts from video texture generation. Furthermore, we demonstrate how the presented technique can be extended to image-based inputs without much effort.

6.2 Our Contributions

Existing approaches either are limited to re-sequencing large amounts of existing image/video data, or to interpolating only vector based drawings.

Our new approach, however, is based on data creation and starts with just a few keyframes (less than 10 frames) drawn by an animator. These frames serve as the transition points required to rearrange generated in-betweening animations as automatically desired or interactively commanded by the animator. By creating in-between frames for meaningful pairs of keyframes, the corresponding transitions remain smooth. The main improvements our work contributes to the automated 2D animation process can be summarized as follows:

- The creation of perpetual cartoon animations, from a very small set of given keyframes (based on vector-based drawings or on image data), displaying content that exceeds the input data;
- The combination of re-sequencing existing material with the automatic generation of new, in-betweening data;
- A simple distance metric applied to estimate the amount of in-betweening frames required two bridge pairs of given keyframes and to prevent useless or meaningless transitions between dissimilar keyframes;
- A quick, iterative and automated animation pipeline, allowing the animator to refine the intermediate results according to his/her expectations;
- A goal-based method, responding to the inputs of the animator, to generate new, user-controlled animations, interactively.

For clarity, this chapter is divided into the following topics. First, in Section 6.3 we give an overview of earlier techniques applied for the creation

---

1 This chapter is based on “Uniting Cartoon Textures with Computer Assisted Animation”, [Van Haevre 05a].
of cartoon animations. Next, in Section 6.4 we present the main parts of our new approach. In Section 6.5 we give an overview of the animation pipeline used for the creation of our targeted animations. Section 6.6 examines the most important steps from the analysis part of this pipeline, and brings us to Section 6.7 where we describe how new animations are synthesized from the available data. We end with some clarifying results (Section 6.8) and a short discussion on the presented technique (Section 6.9).

6.3 Related Work

Employing existing computer-assisted 2D animation approaches, animators roughly can choose between two extreme options. They either have to create animations in the traditional way (e.g., frame by frame [Blair 94, Patterson 94]) or in a fully automatic way (e.g., revert to video textures techniques [Schödl 00, Campbell 02, de Juan 04]).

The latter category delivers satisfying results but heavily depends on large incremental data sets and cannot cope with visual discontinuities. The first, on the other hand, is based on basic interpolating schemes and requires only a small set of input data. However, the amount of interaction requested from the user is often tedious and time-consuming.

One of the most obvious methods for generating new animations is creating in-between frames by interpolating between two or more given frames. In computer-assisted animation, there are essentially two types of interpolating systems: shape-based [Burtynk 71, Reeves 81, Sederberg 92, Kort 02] and skeleton-based [Burtynk 76, Shapira 95, Sederberg 93]. However, in-betweening in traditional animation, is not just interpolating between handdrawn key drawings [Catmull 78]. When drawing the in-betweens, animators utilize their background knowledge of the physical rules of the world, their expert knowledge of when to bend or ignore these rules, and the emotions they intend to evoke by the animation.

In 2002, Bregler et al. used capturing and re-targeting techniques to track the motion from traditionally animated cartoons and assign them onto new 2D drawings [Bregler 02]. By using animation as the source, similar, new animations could be generated. This approach led to very impressive results, but unfortunately several drawbacks prevented it from being used extensively.

The re-targeting process was very dependent on a good choice of the source and target key-shapes which one had to select and draw manually. Further-
more, the animator needed to watch carefully that the chosen key-shapes covered the entire cartoon space (the entire range of possible poses).

Rose et al. presented an inverse-kinematics methodology exploiting the interpolation of example-based motions and positions [Rose III 01]. The key issue of their system was to allow an artist’s influence to play a major role in ensuring that the system always generated plausible results. Starting from a small number of example motions and positions, an infinite number of interpolated motions between and around these examples were generated. This methodology was highly focused on positioning articulated figures and therefore did not lend itself to traditional 2D animation.

Concerning the reuse of motion data, Kovar et al. introduced Motion Graphs to control realistic motion through a database of motion captured data [Kovar 02]. A motion graph is a constructed graph encapsulating connections among different pieces of motion. Graph walks (i.e. motions) are then simply constructed by walking on the motion graph. Kovar’s motion graph technique plays an important role in our animation technique.

As a followup work, Gleicher et al. [Gleicher 03] introduced Snap-Together Motion, which preprocessed a large corpus of motion capture examples into a set of short clips that could be “snapped together” to make continuous streams of motion at run time. The result was a simple graph structure that facilitated efficient planning of character motions. A user-guided process was able to select common character poses and the system automatically synthesized multi-way transitions connecting these poses. The resulting technique was successfully applied to the animation of simple 3D characters in virtual environments.

Similar work was presented by Arikan and Forsyth, focussing on the creation of plausible human-like motions derived from a large set of given motion clips [Arikan 02].

Our inspiration for research on the creation of cartoon textures resulted from the promising work done by de Juan et al. [de Juan 04] on this subject, earlier described in Chapter 5.

6.4 Approach

From video textures creation, we adopt the idea of using a distance metric to estimate the similarity of all pairs of keyframes provided by an animator.
Based on computer-assisted traditional animation techniques, in-betweening is employed to extract new data from the input, according to these calculated distances.

The given set of keyframes is then used to automatically generate sets of in-betweening frames. Based on automatic keyframe in-betweening, incremental changes are made to the data resulting in smooth transitions without visual discontinuities. Furthermore, no preprocessing or postprocessing is required to improve the quality of the resulting animations.

By re-sequencing the input data and synthesizing new art from it, a large amount of new animation data is provided to the animator. The actual amount of in-betweens required, depends on a distance metric preventing possible visual discontinuities. More complex metrics to estimate good transition lengths between the different keyframes have been considered [Wang 03, Wang 04], but are not required due to the simplicity of the depicted content.

Finally, a concept similar to motion graphs [Kovar 02] is applied to control the motion through our data. An optimized cost graph is derived from the generated frames, indicating for all keyframes how many steps are required to travel from one keyframe to another. From the constructed graph, encapsulating the connections among different pieces of motion, graph walks are extracted in a goal based manner by rearranging the generated sets of in-betweens, resulting in interactive animations that incorporate user specified constraints.

6.5 The Data Path

Creating new animations from a small set of input data requires several steps to be executed one after the other which we summarize into two large phases:

1. Analyze the input data and extract new data from it.

2. Employ this data to create new animations by re-sequencing the generated material according to specific rules and animator interactions.

The full pipeline to create a cartoon animation consists of the following steps (see Figure 6.1):

**PHASE 1: Analysis & Data Generation**

1. The animator draws a few basic keyframes (see Section 6.6.1):
Figure 6.1: The pipeline of our process to create endless, or repetitive animations. Phase 1 consists of the analysis of a few input keyframes and generates the required in-betweens to bridge them. Phase 2 allows for the creation of animations which can even be extended to comply with animator interactions.
PHASE 2: Animation Synthesis

5. Random animations are synthesized from the generated cartoon data, by rearranging the generated sets of in-betweens (see Section 6.7.1);

6. A cost graph is derived from the generated frames indicating how many steps are required to travel from one keyframe to another (see Section 6.6.5);

7. This cost graph is optimized to represent the cost that is minimally required to travel between any pair of keyframes (see Section 6.6.6);

8. Using the interpolated frames and the optimized cost graph, interactive cartoon animations can be synthesized (see Section 6.7.2).

Steps 1 to 4 (also numbered in Figure 6.1) suffice to allow the user to create random animations (step 5). When steps 6 and 7 are added to this process, interactions with the generated animations can be incorporated (step 8). In the following sections each step of the presented pipeline is explained in more detail.

6.6 Creating Data

To relieve the animator from the very time-consuming work that is involved in generating sufficient frames for a new animation, a computer aided alternative is required. Starting from a few keyframes, drawn by the artist, a set of new frames should be generated automatically.
6.6.1 Providing the Initial Keyframes

The first step in our animation pipeline consists of the creation of a few initial keyframes. To this end, we employ user-controlled structured 2D modeling and animation techniques [Di Fiore 01] in combination with automatic in-betweening. This methodology clearly distinguishes between a separate modeling and an animation phase, similar to the 3D animation process and has been proven to be very useful for the purpose of creating convincing 3D-like animations, starting from pure 2D drawings, while preserving the artist’s personal style.

Considering 2D animation from a technical viewpoint, two different categories can be distinguished:

- transformations in a plane parallel to the drawing canvas (the XY plane), and
- transformations outside the drawing plane, especially all rotations around an axis different from the Z-axis.

The former category of transformations is relatively easy to deal with, whereas the latter is the main cause of all the trouble in automating the in-betweening process (i.e. the underlying sub-problems of silhouette changes as well as self-occlusion). It is in the latter type of animation where the 3D structure comes into play that is underlying the objects and characters in traditional animation (and which is present in the animator’s – and viewer’s – mind), but which is not present in the 2D drawings.

To tackle this without introducing too much 3D information, Di Fiore et al. developed a solution based on structured 2D modeling and animation techniques [Di Fiore 01]. This was implemented as a multi-layered system. At level 0, objects were modeled as sets of depth-ordered 2D drawing primitives (e.g., subdivision curves or subdivision surfaces). Level 1 managed and processed explicit 2D modeling information and was fundamental in realizing transformations outside the drawing plane: for each set of “important” XY-rotations of the object relative to the virtual camera, the animator created a set of ordered 2D primitives. These were functionally comparable to the extreme frames in traditional animation [Blair 94, Patterson 94].

Level 2 incorporated 3D information by means of 3D skeletons or approximate 3D objects, while level 3 offered the opportunity to include high-level tools.
Multi-level 2D strokes, interpolation techniques and on-the-fly resorting were used to create convincing 3D-like animations starting from pure 2D information. A rigid 3D look was avoided through varying line thickness and the ability to have subtle outline changes that are either impossible or tricky to achieve utilizing 3D models (see Figure 6.2).

Besides “drawing” keyframes, our animation system also includes the possibility to create keyframes by incorporating real images depicting extreme poses. To this end we provided a tool which allows the animator to define a layered mesh structure over certain image parts that contain interesting information. This mesh is constructed from meaningful feature points in the image, as shown in Figure 6.3. This example depicts two extreme poses of a human face and a frame generated between them. During the animation, in-between images of these “real” keyframes are constructed by warping the meshes imposed on the extreme frames to each other in the same order as defined by the layered structure. The remainder of the algorithm is identical to the vector-based version.

This structured 2D approach (i.e. explicit 2D modeling and automatic in-betweening) is the computerized version of the animator’s work place. Unlike purely 3D-based approaches, the resulting animations still have many lively aspects akin to 2D animation. Depending on the purpose of the resulting animation and the quality that is required, the amount of keyframes can reach from just a few to several dozens. In general less than 10 keyframes suffice to cover the usual viewpoints which occur in cartoon space.

As an example throughout the remainder of this chapter we consider the 5 initial keyframes from a simple flower animation depicted in Figure 6.5. Most of the calculations demonstrated in this chapter, relate to these 5 images. This example is, however, very simple and is not chosen to reflect the possibilities of the technique, but rather to clarify the mathematical examples from the text.

6.6.2 Estimating the Pairwise Keyframe Differences

Given a sequence of user specified keyframes, in-betweens should be generated automatically. An important aspect of this process is the calculation of the amount of in-betweens needed. To determine this number, a distance value $d$ between all pairs of frames is calculated. This distance is defined by a function of the displacement of the control points from which the frames are
Figure 6.2: (a. and c.) Two extreme poses of a drawn flower using subdivision curves as drawing primitives (see depicted control points); (b.) An in-between frame generated from them.

Figure 6.3: (a. and c.) Two pictures depicting extreme poses of a human face; (b.) An in-between frame generated from them, using subdivision meshes.
constructed. For each pair of extreme frames the following rules need to be considered:

1. The amount of in-betweens needed between two keyframes depends on the properties of the deformation of the displayed object. If the deformation is global (the magnitude of the largest displacement of any control point remains below a factor $x$ times the average movement of all control points), a sufficient amount of in-betweens should be introduced to minimally interpolate the average deformation. If the deformation is local (only specific parts of the object are changed or the largest displacement is minimally $x$ times larger than the average control point transformation), the largest deformation defines the required amount of in-betweens.

2. This amount is also relative to the magnitude of the deformation of the displayed object. Small changes to the displayed object require only a few interpolation steps, large deformations on the other hand, need a larger number of in-betweens to make sure the transitions are smooth and meaningful. When the extreme frames are too different from each other, interpolation is prevented and an alternative way to connect the keyframes is required (see Section 6.6.6).

For each pair of keyframes $i$ and $j$, the pixel-based distance between the displayed data is measured using the actual positional information of the control points from which the image was built. By averaging all control point deformations per pair, while keeping track of the largest control point displacement, the previously mentioned rules can be taken into consideration. As an example, the flower keyframes from Figure 6.5 represent typical global motions (the largest control point displacement is only a small factor larger than the average motion). As a result, for this specific case most in-betweenings are based on the average distance between their control points.

The selected distance values can be stored in a $n \times n$ matrix $D$ with $n$ the number of keyframes. Each matrix entry $D_{i,j}$ represents the distance from keyframe $i$ to keyframe $j$ (where $D_{i,i} = 0$, $\forall i$), hence, representing their difference in frame content. Matrix $D$ in Equation 6.1 depicts the distance graph corresponding to the 5 keyframes shown in Figure 6.5.
92 Cartoon Animation Based on the Video Texture Principle

\[
D = \begin{bmatrix}
0 & 102 & 189 & \text{max} & \text{max} \\
102 & 0 & 97 & 180 & \text{max} \\
189 & 97 & 0 & 89 & 216 \\
\text{max} & 180 & 89 & 0 & 130 \\
\text{max} & \text{max} & 216 & 130 & 0
\end{bmatrix}
\]  \tag{6.1}

A value denoted by $\text{max}$, indicates that no in-betweens should be generated between the corresponding keyframes because there was not enough similarity detected (a large $d$ value, exceeding the user specified threshold $t$), making automatic interpolation meaningless.

### 6.6.3 Automatic In-betweening

Provided with the user specified keyframes and the distance matrix $D$, the actual interpolations of the keyframes can be performed resulting in the required amount of in-betweens to accomplish a smooth transition from one frame to another. As a global property of the interpolation process, the animator can decide on the amount of interpolation steps used to bridge a specific distance.

At this stage in the pipeline, for each pair of keyframes with a valid frame distance, a short animation sequence is created connecting them. Storing these sets of interpolating frames into a matrix $F$, allows rapid retrieval of the correct frame sequences while generating a new animation. At each matrix position $F_{i,j}$ (zero based), the indices of the frames created between keyframe $i$ and keyframe $j$ are stored. The content of matrix $F$ can be interpreted as a simplified motion graph [Kovar 02], built from the keyframe indices $i$ and $j$, and connected through the generated sequences that bridge them.

Matrix $F$ in Equation 6.2 shows the frame matrix corresponding to the flowers depicted in Figure 6.5. To direct the animation from keyframe 1 to keyframe 3, for example, the in-between frames 79 till 113 need to be displayed, ending with frame 3 itself.

\[
F = \begin{bmatrix}
\text{max} & 5, \ldots, 23 & 24, \ldots, 60 & \text{max} & 60, \ldots, 24 & 78, \ldots, 61 \\
23, \ldots, 5 & \text{max} & 61, \ldots, 78 & 79, \ldots, 113 & 61, \ldots, 24 & 78, \ldots, 61 \\
60, \ldots, 24 & 78, \ldots, 61 & \text{max} & 114, \ldots, 130 & \text{max} & 113, \ldots, 79 \\
\text{max} & 114, \ldots, 130 & 130, \ldots, 114 & \text{max} & \text{max} & 173, \ldots, 197 \\
\text{max} & \text{max} & 172, \ldots, 131 & 197, \ldots, 173 & \text{max} & \text{max}
\end{bmatrix}
\]  \tag{6.2}
Towards an Iterative Process

After the creation of the frame matrix $F$, the first stage in the creation of a new animation is finished. Section 6.7.1 explains how at this point in the pipeline random animations can be generated using a straightforward algorithm.

However, transitions between two keyframes are sometimes not possible because the required in-betweens are absent. If the distance matrix $D$ contains insufficient entries, the animator should consider drawing more frames.

Using the method of automatic in-betweening, an iterative process can be established to increase the amount of initial keyframes. By forcing the interpolation of one or a few extra keyframes between frames with a large relative distance, large displacements of control points can be broken into smaller steps. Furthermore, the animator is always in control and hence allowed to modify these generated in-betweens to his/her specific desires.

Otherwise, an alternative path to travel between these frames, making use of other reachable keyframes, is required. This alternative route, should connect the keyframes in a straightforward manner without deviating too much from the planned animation sequence. To accomplish this, a cost graph is added to the animation pipeline.

Deriving the Cost Graph

At this point in the animation pipeline, a computer generated number of interpolated frames have been created between several pairs of original keyframes. The cost to travel from frame $i$ to frame $j$ can be defined as the length of the frame sequence required to reach $j$. Using matrix $F$, these costs can be stored in a $n \times n$ matrix $C$ where $n$ indicates the number of keyframes.

Each matrix entry $C_{i,j}$ represents the cost to travel from keyframe $i$ to keyframe $j$ (where $C_{i,i} = 0, \forall i$). Matrix $C$ in Equation 6.3 depicts the cost matrix corresponding to the 5 keyframes shown in Figure 6.5.

$$C = \begin{bmatrix}
0 & 20 & 38 & max & max \\
20 & 0 & 19 & 36 & max \\
38 & 19 & 0 & 18 & 43 \\
max & 36 & 18 & 0 & 26 \\
max & max & 43 & 26 & 0
\end{bmatrix} \quad (6.3)$$

This cost matrix $C$ can be interpreted as a cost graph $G$, containing 1 node for each keyframe and edges between all pairs of keyframes for which in-betweens are generated. The edge between frame $i$ and frame $j$ is labeled
Figure 6.4: The graphical interpretation of the cost matrix $C$ from Equation 6.3. Each node represents one initial keyframe; the edge between two nodes $i$ and $j$ is labeled with the cost value $C_{i,j}$.

with cost value $C_{i,j}$. The cost graph corresponding to matrix $C$ from Equation 6.3 is illustrated in Figure 6.4.

6.6.6 Optimizing the Cost graph

The cost graph, however, is not fully connected. Several edges between keyframes are still undefined due to frame distance calculations that result in a value laying above the earlier mentioned user specified threshold. These connections, however, can be created iteratively. By means of successive traversals of already existing connections, a path can be found between any two original keyframes (if the cost graph is closed). In addition, several connections can be improved by means of alternative routes in the existing cost graph. To accomplish this, the simple brute force algorithm of Listing 6.1 is used. It consists of the following major steps:

1. Label positions with an undefined cost value with a maximum cost;

2. For each pair of keyframe indices $(i, j)$, search for an index $t$ for which:
   $C_{i,t} + C_{t,j} \leq C_{i,j}$; if such a value $t$ is found, store $C_{i,t} + C_{t,j}$ as the new cost to travel from keyframe $i$ towards keyframe $j$ and mark keyframe $t$ as a new possible frame to go to from frame $i$ when traveling towards frame $j$;

3. Repeat step 2 until no more improvements are encountered.

The improved cost matrix $C$ indicates for each pair of frames $i$ and $j$ the minimal cost to connect them. Matrix 6.4 represents the optimized cost matrix for the flowers example from Figure 6.5.
6.6 Creating Data

<table>
<thead>
<tr>
<th>OptimiseCostGraph( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>label undefined positions with maximum cost;</td>
</tr>
<tr>
<td>while a cost value changed do</td>
</tr>
<tr>
<td>for each matrix position ((i, j)) do</td>
</tr>
<tr>
<td>search an index (t) for which: (c_{i,t} + c_{t,j} \leq c_{i,j});</td>
</tr>
<tr>
<td>when found: update (c_{i,j}) and store (t);</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>next</td>
</tr>
</tbody>
</table>

Listing 6.1: The algorithm to optimize the initialized cost matrix \(C\) in order to represent the lowest cost between all pairs of keyframes.

\[
C_{\text{optimized}} = \begin{bmatrix}
0 & 20 & 38 & 56 & 81 \\
20 & 0 & 19 & 36 & 62 \\
38 & 19 & 0 & 18 & 43 \\
56 & 36 & 18 & 0 & 26 \\
81 & 62 & 43 & 26 & 0
\end{bmatrix} \quad (6.4)
\]

Encountered deviations from the original paths, improving the cost to travel between two frames, can be stored in a matrix \(P\). At each matrix position \(P_{i,j}\) a set of frame numbers is constructed, indicating which keyframes to aim for, when traveling from keyframe \(i\) to keyframe \(j\). Whenever a better path is found, the current list is cleared and a new set of better deviations is built up.

When no further optimizations are encountered, each element from the set of indices at matrix position \(P_{i,j}\) represents the first node to aim for in a shortest path from keyframe \(i\) to keyframe \(j\). This information can now be used to travel between keyframes using a minimal number of frames. The resulting matrix \(P\) for the flowers example from Figure 6.5 is shown in Equation 6.5.

\[
P = \begin{bmatrix}
- & 1 & 2 & 1,2 & 2 \\
0 & - & 2 & 3 & 2,3 \\
0 & 1 & - & 3 & 4 \\
1,2 & 1 & 2 & - & 4 \\
2 & 2,3 & 2 & 3 & -
\end{bmatrix} \quad (6.5)
\]
6.7 Animation Synthesis

In the following sections two methods are proposed for generating a new cartoon animation from the available data. Both of them are highly related to the concept of a graph walk, introduced by Kovar et al. [Kovar 02].

6.7.1 Creating Random Animations

The first method simply creates a new animation by randomly picking a new keyframe, $j$, as a goal to purchase from the current displayed keyframe $i$. That way, the $C_{i,j}$ frames of the short sequence $F_{i,j}$ that bridges the distance between the current keyframe $i$ and the target $j$ are played. If matrix $F$ doesn’t provide the required in-betweens to connect keyframe $i$ with keyframe $j$, a different random goal is selected.

The resulting animations are completely random and can best be compared with the usability of cartoon textures [de Juan 04]. The keyframes from which they are built serve as the transitions required to rearrange the generated data and to combine the different short animations that exist between the individual keyframe pairs. In most cases, the action represented by the generated cartoon is very simple. More meaningful animations demand a different approach as explained in the next section.

6.7.2 Towards Interactive Animations

Contrary to the random animation creation, we describe an algorithm to generate new animations that take into account interactions by the animator.

Exploiting the cost matrix $C$ in combination with the shortest path info from matrix $P$, animations can be generated interactively. While the algorithm generates random goals for the animation to aim for, the user can interfere and point out the next keyframe to reach for (see Listing 6.2).

For example, while the animation system is displaying one of the in-betweens connecting keyframe $i$ with keyframe $j$, the user can propose a new goal $g$. Several candidates might exist to reach $g$ starting from $j$, all of them through different paths of minimal length (e.g., $P_{4,1}$ for the flower example proposes 2 possible shortest paths: passing through keyframe 2 or through keyframe 3). As soon as keyframe $j$ is reached, one of the elements of matrix position $P_{j,g}$ is randomly chosen as the next intermediate goal to aim for. This new goal is the next keyframe within a shortest path from keyframe $j$ towards keyframe $g$. As long as keyframe $g$ is not reached, new intermediate goals are
6.8 Results

Listing 6.2: The algorithm used to extend the random animation generation of cartoon textures to an interactive approach, creating more meaningful, user directed animations.

introduced by the algorithm, always bringing the playback of the animation closer towards keyframe \( g \). Finally, when \( g \) is reached and no further interactions from the user are recorded, the algorithm continues to pick new random goals (if an endless animation is desired). At any moment the user can make a new request for a specific keyframe, deviating the current graph walk to a new destination.

6.8 Results

In Figure 6.6, a few snapshots are displayed from an animation of a flower, based on the keyframes displayed in Figure 6.5. It took an unskilled animator less than 30 minutes to draw all extreme frames and almost no time (less than a minute) to generate the animation. From the 5 initial keyframes a set of 193 in-betweens was derived, according to the pairwise inter-frame distances. The resulting animation is smooth and the appearance of the flower can be controlled interactively by the animator (e.g., go far left, left, middle, etc.).

By applying a user defined threshold on the allowed inter-frame distances, and using the optimized cost graph, bad transitions are prevented, and a shortest path can be taken to reach the requested flower position. In this example, the in-betweening process was restricted to movements that only bridge keyframes that are located 2 positions (in terms of keyframes) from each other (e.g., from the far left position to the middle one).

Another example of curved based drawings is shown in Figure 6.7. To create this animation, a few snapshots from a moving figure were modeled
from which several are shown in Figure 6.8. In addition, several intermediate frames were created and altered in an extra iterative pass of phase 1 of the presented animation pipeline to control the animated motions. Consequently, our optimized cost graph not only allows for smooth interactive transitions, but also guides the animation through the specifically intended intermediate keyframes. This is accomplished by applying a well chosen threshold on the pairwise frame distances (in this case restricting the automatic in-betweening process to frames with a high similarity). In total, 13 frames were drawn and 473 were added automatically.

Figure 6.10 demonstrates how a video-like result is achieved by applying our algorithm to image data instead of vector based drawings (see Section 6.6.1). The displayed snapshots of this animation are part of a short animation created from a set of 26 keyframes from which a few are displayed in Figure 6.9. The successive smooth changes of the mouth position resemble footage of a speaking person in a convincing way.

This first attempt, based on the mouth positions of the individual letters of the alphabet already proves the usefulness of our technique. Improving this by actually modeling the phonemes of a language should give an even more convincing and more usable result.

A similar example (Figure 6.12) is created from facial expressions. In this case, only the 5 keyframes displayed in Figure 6.11 are used. As an example of the applicability of the resulting interactive animation, this (and the previous) example can be used to extend a standard chat application with smooth facial expression transitions, enriching communication with “realistic” emotions.

6.9 Discussion

In all the given examples, the actual creation of the basic keyframes requires most of the animators’ time. However, this artistic part of the animation process will always remain a task to be performed by the animator himself/herself.

Creating the in-betweens takes only a few minutes while the creation of the animation itself happens in real-time (including possible interactions of the animator). Calculating the pairwise inter-frame distances and the resulting optimized cost graph takes less than a second and is therefore negligible from the rest of the calculations. All examples were tested on a commodity personal computer.
6.9 Discussion

Figure 6.5: The 5 initial keyframes of a simple flower animation (bending from the left to the right).

Figure 6.6: Several snapshots from an animation created from the keyframes in Figure 6.5.

Figure 6.7: Five of thirteen keyframes used to create a moving figure animation. (first 3) 3 modeled poses; (last 2) intermediate keyframes, added by the user in a second pass to control the direction of the animation.

Figure 6.8: Several stills from a cartoon animation created from the keyframes in Figure 6.7. 13 frames were drawn and 473 were generated automatically.
Figure 6.9: Five images out of a set of 26, displaying different placements of the mouth while speaking.

Figure 6.10: Several stills from a video-like animation created from the keyframes in Figure 6.9. The successive smooth changes of the mouth position resemble footage of a speaking person in a convincing way.

Figure 6.11: 5 keyframes used to generate a simple facial expression animation.

Figure 6.12: Several generated frames from the facial expression animation created from the keyframes in Figure 6.11.
We would like to address the issue of controlling the speed of the animation which in general is related to the content of the animation itself. In the current system, the speed of the animation is defined by the inter-keyframe distances and the timing enforced by the animator. Dynamic changes of velocity into an earlier stage of the animation pipeline could be considered.
Cartoon Animation Based on the Video Texture Principle
Our final contribution described in this dissertation concerns the structural realism in tree animations. In Chapter 5 we mentioned how, due to the complex dynamics involved, the employment of our video-based approach fails to create satisfying results. Instead, we propose a more geometrical approach for tree animation, augmented with physically-based motion data.

7.1 Introduction and Motivation

Simulating dynamic natural wind effects on trees remains a challenging task in computer graphics. From an animator’s point of view it is a cumbersome and tedious task to create these effects due to the complexity of the tree shape, the numerous protruding branches and the wide variety of foliage.

Although animation techniques like motion capturing and motion re-targeting have become a very common thing in research, they are impractical to accurately capture difficult-to-model physical movements such as trees moving in the wind.
In addition, many research has been done on the computer-assisted an-
imation of virtual plant models [Mizuguchi 01, Di Giacomo 01, Wesslén 05].
Most techniques, however, focus on the animation of a single plant instance,
controlled by a well defined procedural method or a straightforward physically-
based simulation. Due to their large computational cost, these methods cannot
be applied to multiple plant instances at the same time.

As our final contribution, the upcoming chapter presents a novel method
to create controllable animations of trees by re-sequencing physically-based
acquired motion data from a tree model. Furthermore, the available motion
data can easily be assigned to multiple instances of this tree model in a non-
identical way.

7.2 Our Contributions

The key idea of this research is to capture the motions of one (or just a
few) physically-based tree animations and re-target them onto a large set of
tree instances. By transferring the acquired motion data from a mass-spring
system to new tree instances, larger natural scenes can be animated at real-
time frame rates. Furthermore, the animator can easily interact with the
scene. To respond to his/her requests, the trees are encouraged to display the
captured data corresponding to their inputs. We summarize our contributions
to this topic, as follows:

- A novel method to create user controllable animations of trees;

- Synthesis of new animations by expanding and re-sequencing physically-
based acquired material, reflecting the realism from the original simula-
tion.

- Real-time animation of up to 100 fully animated tree skeletons, derived
  from the simulated motion data.

- An intuitive method for the animator to direct the animation of the trees
  at each arbitrary moment.

- A 2D visualization of the tree shapes, based on a cartoon style.

\[^1\text{This chapter is based on “Physically-based Driven Tree Animations”, }\text{Van Haevre 06.}\]
Even though our method is a skeleton-based technique, the final item in this overview indicates how we also paid some attention to the visualization of the animated trees. For representation purposes, a simple technique will be used, based on the assignment of a simple shape to each three dimensional skeleton element. These shapes can easily follow the movements defined by the skeletal joints.

To visualize these skeleton-based trees in 2D (e.g., a cartoon style), an articulated skeletal structure is used [Di Fiore 03].

First, we summarize the content of the rest of this chapter. After introducing work related to ours in Section 7.3 and describing the main parts of our approach in Section 7.4, the data path to be followed to generate new non-identical animations is introduced (Section 7.5). Next, the central stages of this data path are elaborated. Sections 7.6 and 7.7 focus on the gathering and processing of motion data, whereas Section 7.8 concentrates on the synthesis of new animations based on this data. Section 7.9 provides clarifying results. We end with a short discussion in Section 7.10.

### 7.3 Work Related to Ours

In 2D, the most obvious method for generating animations is creating keyframes and their interpolating in-between frames. Because in our final technique trees are represented as 3D geometries, we focus on three dimensional skeleton-based in-betweening techniques [Burtnyk 76, Shapira 95, Sederberg 93].

During the animation process there is the issue of how to control the skeleton. It can be determined by motion capture or manual posing but in the case of tree skeletons this becomes practically impossible to do.

As a result, many different techniques have been investigated to animate trees, often based on a physically-based representation of their models. The next few paragraphs elucidate on some of the work presented earlier that aimed for the creation of plausible tree animations.

Already in the early nineties, research was done by Shinya and Fournier on a good representation for the forces that influence tree models in a virtual environment [Shinya 92]. Even though their stochastic or physically-based wind motions can easily be imported in our work the representation of correct wind fields is not considered as a goal in our research and replaced by simple motions, interactively defined by the animator. Hence, the animator remains in full control over the motions depicted.
Physically-based Driven Tree Animations

Based on Shinya’s wind model and algorithms imported from wind engineering, Setas et al. described a method to generate a wind velocity field efficiently [Setas 95]. From these velocities, wind forces were derived which could influence vegetation present in a virtual scene.

Stam too described a method to simulate the motions of tree-branches subjected to turbulence [Stam 97]. Similar to the method we introduce in Chapter 7, a physically-based simulation of the tree motions was performed and the displacements encountered during this simulation were stored. Afterwards, these displacements could be assigned to many instances of the same tree model. However, in general the depicted animations remained visually identical.

Sakaguchi and Ohya started from an image-based tree modeling method and added dynamics to them by applying a simple explicit integration method on the motions of the connected branch-segments [Sakaguchi 99]. The simplicity of the method, however, required the selection of good parameters to prevent the mathematical system from collapsing.

A similar method was described by Wu et al. [Wu 99].

In 2001, Mizuguchi et al. introduced Move Trees which are graph structures representing connections in a database of motion [Mizuguchi 01]. This technique is widely used in the gaming industry as it meets the requirements of online motion generation. Unfortunately, motion trees need to be created manually which remains a cumbersome task.

Regarding data-driven animation, Perbet and Cani used, in the same year, the procedural animation of a single, physics-based, deformable, grass mesh primitive (with texture-based level of detail), to animate massive grass prairies at interactive rates [Perbet 01].

Di Giacomo et al. animated small forest scenes with up to 256 moving trees, focussing their research on game play purposes [Di Giacomo 01, Di Giacomo 03]. The trees were built procedurally from a compact geometrical data set, and their motions resulted from the combination of two animation approaches: a procedural method used to animate the trees at different levels of detail in order to model the action of wind, and a physically-based method allowing user interaction with the trees. Their main contribution was to combine both approaches, so that physically-based animation was used only when actually needed (e.g., to process user interactions with trees at close range) and could be discarded afterwards. Furthermore smooth transitions between
the different animation methods were performed to avoid “popping” effects. When focussing on interactive game play, this semi-procedural method produces acceptable results.

In 2004, a method was proposed by Ota et al. to mimic the natural motions of individual leaves and branches swaying in a wind field in real-time without using time-consuming physical simulation techniques based on the equations of motion [Ota 04]. The method used a hybrid approach combining a stochastic method and a simulation method. The stochastic method was based on $1/f^\beta$ noise, which can be observed in various natural phenomena, and provided natural motions to leaves and branches. In addition, a simple simulation method based on the spring model was applied to the branches to enhance the reality of their motions.

However, to create for each different type of tree the desired tree motions, a trial and error approach was required to modify the animation parameters, making the technique less interesting for the animation of large-scale scenes.

Beaudoin and Keyser presented a method for generating and using levels of detail in the simulation of plant motion [Beaudoin 04]. They created a plant model and from its initial representation they automatically created a simplification hierarchy that, when simulating plant/wind interaction, behaved the same as the original. This allowed simulation of large groups of plants because the hidden or far away items could be significantly simplified, yet yielded visually realistic results. The simulation was stored as a simple articulated tree structure (generated, for example, by an L-System). The simplified simulation levels of detail were created by combining either sets of child branches or parent-child connections, with bearing in mind guaranteed error bounds. Furthermore, smooth transitions between the simulation levels of detail were provided at run-time.

Wesslén and Seipel described a straightforward technique for the animation of swaying stems and fluttering foliage, executed locally on a graphics processor [Wesslén 05].

Swaying was simulated by a periodical rotational motion around the origin of the trunk, with the rotation amount relative to the distance to the base of the tree (i.e. a tree stem was considered to be an upside down pendulum). Fluttering leaves were animated by a quick rotation around the spine of the leaf. The swaying motion, however, was uniform over the tree, so branches did not move independently of each other.
Physically-based Driven Tree Animations

Their work allowed for real-time animation of moderately sized areas of forest, but due to the simple heuristic animation techniques implemented, the credibility of the synthesized motions still remained low.

Most research mentioned above focussed on simulating the movement of plants with thin or small leaves, where each leave is modeled as a single polygon. In contrast, Wu et al. came up with a technique to simulate the realistic movement of broad-leaf plants [Wu 06]. These plants require a different kind of simulation due to the larger response these leaves have to external forces.

By modeling the stems and branches using a simple segment structure, and representing the leaves with a grid of interconnected mass points (inspired by cloth simulation) a physically-based technique was developed. Each mass in the system responded to external forces which were propagated through the entire model. Real-time frame rates were achieved for only a single plant model, making the technique impractical for large-scale scenes containing multiple tree instances.

In 2006, Zhang et al. presented a quasi-physically based approach for the interactive simulation of large-scale dynamic forest scenes under different wind conditions [Zhang 06]. The wind field was represented as a stationary stochastic process and geometrical complexity was reduced by adopting a hybrid geometry/image representation scheme to model the appearance of the trees without sacrificing too much image quality. However, simplified mechanical rules were employed to compute the movements of the tree models.

A recent publication on data-driven animation, related to our work, was made by James et al. [James 06]. They introduced Mesh Ensemble Motion Graphs, a precomputation-based approach for data-driven animation of high-dimensional, deformable mesh ensembles that exploits asynchronous motion. They illustrated how asynchronous transitions can enable the reuse of modest amounts of expensive, precomputed simulation data for high-quality mesh animation in real-time contexts.

7.4 Approach

Technically the challenge is to capture, extend and re-sequence an existing animation. To this end, our approach borrows several ideas from motion synthesis, computer-assisted animation and motion graphs. It combines re-sequencing of existing material with the automatic generation of new data.
Furthermore, the animator can direct the animation at each arbitrary moment using a goal based motion algorithm.

First, a small set of motion data is gathered from a physically-based driven tree animation. Next, an optimized motion graph is constructed from the acquired data indicating all possible transitions from one tree pose to another. By creating in-between frames for all pairs of keyframes we ensure smooth transitions. Finally, by walking on the motion graph new non-identical animations are synthesized.

The resulting animations are smooth, controllable by the animator and suitable for different production targets including 3D virtual environments (e.g., games) and 2D stylized animation.

7.5 The Data Path

Figure 7.1 depicts a schematic overview of the main parts of the data path, consisting of 3 phases:

- Phase 1 (motion gathering) performs an acquisition step to gather animation data from a small set of physically-based tree models;
- Phase 2 (motion processing and optimization) extends this captured data into a motion graph;
- Phase 3 (motion synthesis) synthesizes new animations.

First we give an overview of these different steps. In the following sections, we will explain them into more detail.

PHASE 1: Motion Gathering

1. A tree skeleton is created or acquired;
2. A physically-based representation of the tree is constructed;
3. External forces are exerted onto the model (e.g., wind, gravity, etc.);
4. The trajectory of the physically-based motion is captured.
110 Physically-based Driven Tree Animations

PHASE 2: Motion Processing and Optimization

5. The basic motion path is recreated (i.e. the captured animation is reconstructed);

6. A directed motion graph is constructed, extending the captured trajectory;

7. The resulting motion graph is optimized by reducing the amount of path cycles, by avoiding possible dead ends, etc.

PHASE 3: Motion Synthesis

8. One or more new trees are instantiated in the scene;

9. An optimized motion graph is attributed to each instance;

10. User inputs are processed;

11. The scene is rendered according to the targeted production (3D polygon rendering, ray tracing, 2D stylized rendering, etc.).
7.6 Gathering Motion Data

The first step in the presented data path consists of data acquisition.

7.6.1 Creating a Simple Tree Skeleton

First of all the actual tree models that are to appear in the rendered scene need to be provided. Many instances of each of these reference models will be included in the final animation.

Earlier research already focused on different methods that allow the creation of realistic tree models including the use of L-Systems [Lindenmayer 68, Prusinkiewicz 90, Prusinkiewicz 97], applying a component based technique to describe the topology of the plant [Lintermann 96, Deussen 97, Lintermann 99], or actually modeling the tree structure using a professional modeling toolkit [Aitken 03]. More information on this subject can be found in Part I of this dissertation.

For research purposes we focus on the creation of tree skeletons based on environment sensitive L-Systems using the techniques explained in Chapters 3 and 4 of this document [Van Haevre 03, Van Haevre 04b]. These models are influenced by the availability of light and growth area and reveal realistic growth behavior.

7.6.2 Providing a Physically-based Support System

In this step, we attach a mass-spring system to each reference tree skeleton. This mass-spring system allows for realistically looking, physically-based tree animations and pulls the tree skeleton to a rest state when no external forces are applied to it.

The system consists of masses attached to the tree skeleton joints. These masses are interrelated using linear and angular springs, which both restrict the motion of the individual parts in different ways. Constraints can be formulated by means of a set of behavior functions: the linear springs minimize the length of a single branch segment to a specified rest value (Equation 7.1), while the angular springs constrain the angle between adjacent branch parts, thus restricting the motion of a branch to a rest angle with respect to its parent branch (Equation 7.2). In both equations $x_{ij}$ equals $(x_i - x_j)$.

\[
C_{\text{linear}} = |x_{ij}| - L \quad (7.1)
\]

\[
C_{\text{angular}} = x_{ij} \cdot x_{kj} - |x_{ij}||x_{kj}| \cos(\alpha) \quad (7.2)
\]
Minimizing $C_{linear}$ is equivalent to reducing the difference between the distance from branch joint $x_i$ to $x_j$, and their rest distance $L$. $C_{angular}$ is defined by the definition of the inner product, based on the rest angle $\alpha$ within the triangle formed by branch segments $x_{ij}$ and $x_{kj}$. A fixed rest angle $\alpha$ requires the positions of the joints $i$, $j$ and $k$ to shift until $C_{angular}$ is minimized.

These behavior functions are translated into force laws based on Equation 7.3 (we refer the interested reader to [Witkin 01] for an in-depth explanation). The forces act on the tree joint masses and enforce the correct branch configuration when external forces are applied to the system.

\[
F_i = (-k_s C - k_d \dot{C}) \frac{\partial C}{\partial x_i}
\]  

(7.3)

The parameters $k_s$ and $k_d$ correspond to the spring stiffness and damping constants. The linear spring forces acquire large stiffness values and damping constants to prevent the branch segments from changing in length. The angular springs require much smaller values, depending on their position in the skeleton. When located high in the tree structure, bending occurs much more easily, and less stiffness and damping is required.

The weight of each mass should withstand all spring forces induced by its offspring. This implies how for each joint the weight attached to its mass should be larger than the sum of the weights of its descendants. This way, the mass-spring system is able to endure the complete weight of the tree.

The mass-spring system’s behavior is evaluated by means of implicit Euler steps. Each step results in a linear system of equations and is solved using the conjugate gradient method. Both techniques were implemented as explained by Baraff [Baraff 98] and Witkin [Witkin 01].

Figure 7.2 shows an example of a tree skeleton with a mass-spring system attached to it. This configuration is influenced by a wind force causing the skeleton to bend towards the right side.

### 7.6.3 Capturing the Animation Trajectory

The final step in the data acquisition phase is to extract meaningful animation data from the previously described mass-spring system.

To capture the tree motions, samples are taken from the branch joint positions at equal time intervals. This information is stored together with the corresponding velocities and the forces applied to them at the time of recording. For each tree skeleton, a few of these trajectories can be recorded, containing similar or alternative tree behavior.
7.7 Motion Processing and Optimization

Given the captured data, we want to reproduce meaningful motions in a computational inexpensive way. The following subsections describe how a motion graph can be constructed starting from a simple motion path. This graph is constructed for only one specific control point in the tree. This point is used to guide the tree motion so it reflects the user’s intended directions.

We empirically found that the choice for the control point can best be limited to the set of points located high in the tree structure as they provide the best results during the construction of the data structures we will define later.

7.7.1 Constructing the Basic Animation Path

Our first step in constructing the motion graph is providing the basic animation path from which the motion graph can be derived. This backbone consists of one motion path assembled from all position samples from the control point. For each pair of successive position samples a path segment is added to the backbone (see Figure 7.3), by means of a short Bézier curve.

At this point, the resulting motion path is limited in its possibilities. It
114 Physically-based Driven Tree Animations

Figure 7.3: A top view of the motion backbone, assembled from all position samples from one control point. For each pair of successive position samples (i.e. blue dots) a path segment (i.e. blue curve) is added to the backbone by means of a short Bézier curve.

only allows for the playback of the originally captured motion and results in a dead end. For these reasons, we need to extend this backbone to a more useful graph with greater connectivity, using transitions. These transitions are short path segments connecting more distant parts of the original motion, allowing the motion generating algorithm discussed in Section 7.8.1 to output new movements and prevent the animation from reaching a dead end.

7.7.2 Towards a Motion Graph

Deviating from the current motion path is not as trivial as it seems. Even though the motion path crosses itself at several places within motion space, the tree pose can change drastically when “taking a turn” at a crossing. To
Figure 7.4: When two path segments $P$ and $S$ cross, a transition can be created, linking the two motions represented by both segments. If sample $p_t$ has two nearest neighbors $s_{i+1}$ and $s_{i+2}$ lying forward with respect to its position, $s_{i+2}$ is accepted as a transitional point resulting in the creation of the smooth path segment from $p_{t-2}$ to $s_{i+4}$.

cope with these differences, a sufficiently smooth interpolation mechanism needs to be applied in order to bridge the differences in tree pose at the start and end location of the transition.

First of all, transition points need to be detected in the motion path. This is achieved by applying a nearest neighbor based algorithm at each position sample while limiting the search to a distance $d$ equal to the distance between the current sample $p_t$ and the next sample on the path $p_{t+1}$ (see Figure 7.4). Only samples located in front of the current sample are retained, because they lie forward with respect to the current samples velocity vector. This nearest neighbor information can be derived in a preprocessing step.

Consider the example shown in Figure 7.4. If sample $p_t$ finds a neighbor at sample $s_{i+2}$, a transitional path is added. This transition is elongated by starting a few samples earlier and ending a few samples later to allow for a smoother interpolation. In this case a transitional path is added to sample
Figure 7.5: A closed directed motion graph. It is derived from the motion backbone of Figure 7.3 (indicated in blue), extended with smooth transitions at each crossing (indicated in green) and contains no dead ends (the pruned part is indicated in red).

$p_{t-2}$, going towards $s_{t+4}$, bridging the differences of the tree skeleton between both sample positions.

Figure 7.5 shows the motion backbone (indicated in blue) of Figure 7.3 extended with smooth transitions at each crossing (indicated in green).

Currently, there are no guarantees that the graph will produce new behavior indefinitely. To prevent the motion graph traversal from reaching a dead end caused by the limited amount of captured samples, the graph requires further updates. Furthermore, the deviations from the test position towards two different nearest neighbors, often result in almost identical transitions. Hence, a few optimizations to the graph construction are required.
7.8 Motion Synthesis

7.7.3 Optimizing the Motion Graph

Traversing the backbone from the end to the start, samples can be removed when they only provide one transition (to the next sample on the path). When a sample is reached containing an additional transition towards a sample located elsewhere in the backbone, we know the beginning of the dead end is found and all samples lying further down the path can be removed. For each sample in the motion graph, possible transitions towards this pruned segment are removed too.

When several neighboring points $s_n$ for one sample point $p_t$ are found, located on the same segment $S$ of the motion path, only those samples $s$ that are located further in time on the original path are kept (e.g., in Figure 7.4 neighbor $s_{i+2}$ is accepted and $s_{i+1}$ is refused because $s_{i+1}$ is succeeded by $s_{i+2}$, which is located further in time in the original motion path). We abandon these extra transitions because they result in an identical behavior.

As a result of these optimizations, a closed directed motion graph is produced, consisting of the original sample connections, smooth transitions at crossings and without dead ends. In Figure 7.5 this pruned dead end is indicated in red. If desired we can further refine the graph by removing local cycles, restricting the amount of transitions, etc.

7.8 Motion Synthesis

After constructing the optimized motion graph we can employ the information it represents to create new, physically-based driven tree animations.

7.8.1 Generating New Animations

For each tree, multiple trajectories of motion can be recorded, all derived from the same mass-spring system. To each tree in the animated scene, first a randomly chosen instance of gathered data is assigned. Next, one point is assigned to each tree, controlling the animation. All this can be done in a preprocessing step.

Random Animations. If no user interaction is required, random animations can be created for each individual tree, simply by traversing the motion graph of its control point.
While interpolating between two sample positions the position of all other tree points is updated according to a similar interpolation, but applied to their own sampled data. Hence, a control point decides the trajectory for all other tree points.

Each time a sample point is reached on the controlling motion path, all possible transitions leading away from this point are eligible for being chosen. Randomly choosing such a transition results in a rearrangement of the original tree movements, providing an overall random tree animation.

**Interactive Animations.** A simple extension to the random method can provide in the specific intentions of an interacting animator or respond to external forces applied to the model.

Instead of randomly choosing a possible transition away from the current sample position, an evaluation of all available transitions is made. The direction the original captured motion path would lead to after taking a transition, is estimated (by looking at a few samples ahead) and compared to the user specified direction. The transition leading the animation best towards the animators intentions is selected. This estimation can also take into account the velocities of the successive samples located each corresponding path.

The use of several alternative motion captures, together with the random selection of a single control point will provide different, but overall similar responses from the trees to the user inputs.

### 7.9 Results

In Figure 7.6, a few snapshots are displayed from 3 animations of individual tree models. Using the real-time mass-spring solver, it took an unskilled animator (like myself) less than 1 minute to provide a sufficient set of motion samples from which the resulting animations were derived. Constructing the corresponding optimized motion graph required less than 1 second and the generation of new animations was done interactively at realtime frame rates. Note, how the animations generated using our system can easily be rendered in a style which is extremely difficult to animate by traditional means. In this case a cartoon-like style [Di Fiore 03] was applied.

Figure 7.7 shows two examples of productions utilizing our technique. The left hand side image demonstrates how our method can be employed within a virtual environment (e.g., a game). On the right hand side a still from a cartoon animation is depicted.
7.9 Results

Figure 7.6: Several snapshots from three new animations, constructed from the same tree model. A hybrid 2D/3D rendering method was applied resulting in a cartoon-like style.
Figure 7.7: Two examples of the applicability of our technique. (left) a 3D virtual environment; (right) a cartoon animation.

As a final example, Figure 7.8 depicts two snapshots of a set of 100 tree skeletons, fully animated at a real-time frame rate. The green arrows indicate wind forces exerted by the user. It can be observed from these images how all trees correspond accordingly.

7.10 Discussion

Our technique can represent up to 100 fully animated, skeleton-based tree models, in real-time, even without the use of algorithms that accelerate the visualization of large object quantities.

Furthermore, the resulting animations correspond to the animator’s interactions and display realistic behavior because the physically-based nature of the gathered motion data is reflected upon the final animation.

However, during the animation process, the whole state of the tree is represented by a single point. This is convenient for interpolation, but might lead to unrealistic behavior if the set of associated states is too uncorrelated.
Figure 7.8: Two images of a set of 100 tree skeletons, fully animated at real-time frame rates. The animator exerts wind forces (depicted by the arrow) to the scene resulting in realtime, non-identical, bending behavior of each individual tree.
8

Concluding Remarks

This dissertation is concluded with a short summary of the different aspects of our work and the achievements they yield. Meanwhile, possible topics for future work are mentioned.

8.1 Summary and Future Directions

We contributed to different aspects of current computer graphics research. Both modeling and animation techniques were addressed, bearing in mind the realistic representation of natural objects like plants and trees.

One of the goals in our work was to extend the structural realism of plants and trees, by incorporating environmental influences in their growth process. Furthermore, we focussed on simplifying the creation of short, perpetual animations and provided video-based and physically-based methods to animate plants and trees.

The Competition for Light and Space. In the first part of this dissertation, we proposed a simple but effective extension to L-Systems taking into
account environment illumination and spatial occupancy. The presented technique was based on an emission of photons, intersecting different tree-parts. Each intersection resulted in an update of a few counters associated with available growth directions. The accumulated counters were used to guide the process of plant part development.

Furthermore we discussed how our approach is based on several ad-hoc assumptions about how plants react to light and how literature on plant morphogenesis should clarify several of them. Secondly, many aspects of the algorithm presented were considered to require some refinements. More advanced irradiance estimation techniques, for example, could be considered to find the predominant illumination direction. In addition, other approaches should be considered to describe plant growth.

**Plant Growth based on Ray Density Estimation.** Following up this first attempt, we presented a new approach, based on a ray density calculation (inspired by Jensen’s photon mapping method [Jensen 01]) to acquire an estimation of the environment illumination by means of a predominant illumination direction. This information was used in a simulation of plant growth, allowing several topological rearrangements of the plants to obtain an optimal growth direction. The flexibility and accuracy of the algorithm, together with its low calculation time and limited memory usage ensured an innovative technique, attractive for plant modeling applications.

Due to randomness in the light emission, our ray density estimation exhibits noise, which produces fluctuations in the growth parameters. This can be reduced by using a different estimator. For future research we would like to consider the nearest neighbors method which always guarantees a sufficient number of rays to do statistics. Secondly, we would like to consider the exploitation of data available from real plant species, and to experiment with new plant representations.

**Video Textures of Plants.** The second part of this document commenced by describing our work on the creation of video textures of plants, based on the symmetric character of their motions. The new technique was presented as an alternative to the traditional method by Schödl [Schödl 00], and exploited the similarity of the surrounding frames at specific video positions, to reverse video playback. Hence, the original data was extended to a new synthesized version, without visual artifacts or unrealistic content.

First, a technique was described to determine the symmetric properties of each frame in a given video sequence. From this information probabilities
were derived, indicating how suitable each frame was to become a turn point for video playback. An adaptive algorithm was presented to synthesize a new endless video stream or video loop, maximally exploiting the available video frames.

Clearly, a hybrid method in which the traditional technique is extended with our algorithm, would increase the randomness of the generated video sequence. The combination of both random transitions and reverses in the video playback should allow for sufficient reorganizations of the original video sequence, while preventing visual inconsistencies.

**Cartoon Animation Based on the Video Texture Principle.** Next, we presented a novel method that facilitates the creation of endless vector-based or image-based 2D animations. To this end, a hybrid approach was applied benefitting from several ideas and techniques from video textures, computer-assisted animation and motion graphs. By combining the re-sequencing of existing material with the automatic generation of new data into a two-phase animation pipeline, new animations could be synthesized. Furthermore, the animator was given the possibility to interfere with the animation process at each arbitrary moment. The resulting animations were smooth, based on very small inputs, and required no postprocessing.

**Physically-based Driven Tree Animations.** As a final contribution, we presented a novel method to create controllable animations of larger sets of trees. Our new approach combined re-sequencing of existing material with the automatic generation of new data. Furthermore, the animator was able to direct the animation at each arbitrary moment using a goal based motion generating algorithm.

The presented method started from a small amount of motion samples, gathered from a few physically-based simulations. Next, an optimized motion graph was constructed from the acquired data indicating all possible transitions from one tree pose to another. By creating in-between frames for all pairs of keyframes we ensured smooth transitions. Finally, by walking on the motion graph new non-identical animations were synthesized for multiple instances of the original tree model.

Results demonstrated how the synthesized animations were smooth, controllable by the animator and suitable for different production targets including 3D virtual environments (e.g., games) and 2D stylized animations.

As an interesting extension to this work, we could consider to gather motion data by means of analyzing a real input video sequence of a tree moving
in the wind, and thus bypassing our physically-based input acquisition. In other words, we would like to synthesize new tree animations by re-sequencing real input video data, similar to the video textures technique [Schödl 00].

Furthermore we are considering to apply our technique to other mass-spring systems including cloth and hair simulation.

Most of the techniques presented in this document are focussed on increasing the perceived realism observed by the user when confronted with virtual plant models and/or their animations. By simulating real-life phenomena like the illumination sensitivity of plants or the responses of branches to a simple wind force, we contributed to both the credibility of visualized natural scenes and the ease to produce them in a convincing way.
Appendices
In this first appendix, we give a brief overview of the most important aspects of L-System based modeling and the way we interpreted their implementation. After defining the basic L-System concept, the most important features of the L-System methodology will be elucidated. We mainly focus on the properties that make up the basis of this research topic and summarize the main features of L-Systems pertinent to the chapters of the first part of this dissertation.

A.1 Defining L-Systems

We define an L-System as a string rewriting system (introduced by the biologist Aristid Lindenmayer [Lindenmayer 72], hence the ‘L’ in L-Systems), similar to a Chomsky grammar, that constructs a generally more complex string of characters from a less complex string by applying production rules.

A production rule describes how to replace a symbol by a new sequence of symbols every time its left-hand side symbol matches with a symbol from the original sequence. These replacements happen in parallel for all occurrences of any left-hand side of a production in the string and are applied iteratively until no occurrences of a left-hand side of a production rule occurs in the
constructed string. Symbols of the string not matching a left-hand side symbol of a production rule are assumed to be operated on by the identity production (i.e. remain unchanged).

In the most extensive case, a production has the format:

\[
id : lc < pred > rc : cond \rightarrow succ : prob
\]

where \( id \) is the production identifier (or label), \( lc, pred, \) and \( rc \) are the left context, the strict predecessor, and the right context, \( cond \) is the condition, \( succ \) is the successor, and \( prob \) is the probability of production application. The strict predecessor and the successor are the only mandatory fields.

Productions may be applied sequentially to one module at a time or they may be applied in parallel with all modules being rewritten simultaneously in every derivation step. Parallel rewriting is more appropriate for the modeling of biological development, since development takes place simultaneously in all parts of an organism.

Simulation starts from a predefined initial structure or axiom, and proceeds in a sequence of discrete derivation steps. In each step, the rewriting rules or productions replace all modules in the predecessor string by successor modules. The applicability of a production can depend on a predecessors context, values of parameters, and on random factors. The resulting developmental sequence can be viewed as the result of a discrete-time simulation of development.

### A.2 Geometric Interpretations of Strings

Starting with the axiom, the L-System replaces each of the characters (in parallel) using a predefined set of production rules. Afterwards, each of the generated symbol sequences acquires a proper visual interpretation, in order to obtain a visual model.

A first method to achieve this, called geometric replacement consists of replacing the individual symbols with geometric elements: lines, corners, \( V \)-shapes, etc.

A different technique to do this, introduced in [Szilard 79] is Turtle Interpretation or Turtle Graphics, where the string of characters acts as a sequence of changes, applicable to the state of a drawing tool (a pen, the “turtle”, \ldots). Each symbol represents a specific change to the state of the pen.

The state of the turtle consists of the turtle position and its orientation in the Cartesian coordinate system, as well as various attribute values, such as current color and width. The position is defined by a vector \( \vec{p} \), and the...
A.2 Geometric Interpretations of Strings

Figure A.1: The turtle’s state is defined by a vector $\vec{p}$, and the orientation is defined by three mutually perpendicular vectors $\vec{H}$, $\vec{L}$, and $\vec{U}$, indicating its heading and the directions to the left and up. (Image source: Prusinkiewicz 97)

Consequently, after the developmental sequence, the generated string acts as a set of commands for a turtle graphics interpreter to generate geometric content.

Symbols like $+$, $-$, $\&$, $\wedge$, $\backslash$ and / rotate the state. Other symbols can move the state to a different position, for instance when drawing a branch. A more complete set of the different symbols used for L-System-based modeling can be found in Prusinkiewicz 97.

One of the most interesting applications of L-Systems is the creation of fractals (which most of the time have a rather unnatural and synthetic visual representation, because of the self-similarity on every recursive level). The fractal displayed in Figure A.2 is based on the L-System described in Section A.3.1 and consists of two steps: first a square is drawn; next, each segment is refined recursively based on one simple rule: $F \rightarrow FF-F+F+F-FF$.

In the next section an in-depth overview is presented of the different types and corresponding properties of the L-Systems used to generate plant models. The taxonomy presented here is based on Rick Parent’s classification Parent 02.
A.3 L-System Taxonomy

We start with the most simple form of L-Systems and iteratively add more computational power to come up with a tool to describe environment sensitive plant growth.

A.3.1 D0L-Systems

The simplest class of L-Systems is deterministic and context-free and are called D0L-Systems. For each symbol in the axiom (the starting sequence) and the strings constructed from it, only one production rule is applicable. The position of the symbol within the original string is not taken into consideration. An example is given below:

\[ \omega : F + F + F + F \]
\[ p_1 : F \to FF - F + F - FF \]

A.3.2 Bracketed L-Systems

The inherently linear interpretation of the D0L-Systems described above can be noticed from these graphical interpretations. To be able to represent the branching patterns in plant models a mechanism needs to be introduced allowing deviations from this linear sequence. In Bracketed L-Systems, brackets are used to mark the beginning and the end of such a deviation, allowing additional offshoots from the main lineage. These brackets correspond to push and pop stack operations and result in temporary storages of the interpreters state.
(i.e. the position and orientation of the current drawing tool). Arbitrarily deep branching is possible and the branching level can be taken into account when required. The following L-System demonstrates a simple bracketed L-System:

\[
\begin{align*}
\omega & : FAF \\
p_1 & : A \rightarrow [+FAF] \\
p_2 & : A \rightarrow F
\end{align*}
\]

### A.3.3 Nondeterministic L-Systems

Important to notice is the non-deterministic property of the previous example. Two production rules are applicable when the symbol \( A \) is encountered, leaving the decision which to pick to the L-System interpreter. The possibility to control this choice brings us to the next type of L-Systems, Stochastic L-Systems.

### A.3.4 Stochastic L-Systems

By adding a user-specified probability to each production rule, we can indicate how likely it is for a certain rule to be applied to a symbol, taking into account that the probabilities assigned to productions with the same left-hand side sum up to one. Consequently, the outcome of the rewriting process becomes unpredictable and a wide range of possible outputs is generated from the same system. This is an important property when modeling plants because it provides the dissimilarity between different plants we can observe from nature. Often, the assignment of a higher value to a specific production rule is reflected in the resulting plant model. For example, giving higher stochastic values to rules that split the state by means of brackets than to rules providing linear growth, results in more distinct splitting within the overall model. This indicates how the use of stochastic rules provides diversity in the outcome of the system, while the creator of the system remains in control of the overall shape of the resulting models. The following L-System demonstrates the stochastic extension to the previous example, resulting in a more controlled form of non-determinism:

\[
\begin{align*}
\omega & : FAF \\
p_1 & : A \rightarrow [+FAF] : 0.8 \\
p_2 & : A \rightarrow F : 0.2
\end{align*}
\]
A.3.5 Context Sensitive L-Systems

The next example shows how the current set of available L-System techniques can be extended to take into account a valid context for the predecessor in order for the production rule to be applicable. As a result, a symbol has different production rules associated with it, depending on the context in which it appears. The context in these context sensitive L-Systems is indicated using a left-hand and/or right-hand side sequence of symbols next to the predecessor (separated by ‘<’ and/or ‘>’):

\[
\begin{align*}
\omega & : FAF \\
p_1 & : F > A \rightarrow A \\
p_2 & : A < A > F \rightarrow F
\end{align*}
\]

The following steps are encountered during the processing of this L-System, starting from its axiom:

\[
\begin{align*}
\omega & \rightarrow FAF \\
& \rightarrow_{p_1} AAF \\
& \rightarrow_{p_2} AFF
\end{align*}
\]

Productions with shorter contexts are usually given precedence over productions with longer contexts when they are both applicable to the same symbol, but this depends on the implementation of the L-System interpreter. In the case where the context of the symbol to be replaced matches with the context of multiple production rules, the earlier mentioned stochastic procedure is applied to select one.

When the required local context of a symbol within a production rule consists of a maximum of \(n\) terms on the left-hand side and \(m\) symbols on the right-hand side, we define the L-System as a \((n, m)\)L-System. If the context is one-sided, the L-Systems are referred to as \(n\)L-Systems, where \(n\) is the number of context symbols taken into account (hence, the ‘0’ in D0L-Systems which are deterministic context-free L-Systems).

A.3.6 Parametric L-Systems

The next refinement to the L-System concept follows from a simple observation. There are two aspects of growth easily modeled using L-Systems: changes in topology and elongation of existing structures. Changes in topology are covered by bracketed L-Systems where branching is encapsulated in productions like \(A \rightarrow F[+F]F\). Elongation on the other hand needs to be handled using specific production rules like \(F \rightarrow FF\) or the combination of
$F_1 \rightarrow F_2$ with $F_2 \rightarrow F_3$. Regrettably, the first solution results in an exponential growth where $F$ represent the smallest unit of growth. The second solution tackles this, but requires a very large set of different $F_i$ symbols to take care of the different lengths encountered during the growth process. Clearly, some sort of parametrization is required to handle this process more smoothly.

In *Parametric L-Systems*, symbols get one or more parameters attached to them indicating additional information like length, thickness, age, color, type, etc. In addition, optional conditional terms can be associated with production rules testing the state of these parameters while adding additional control over the applicability of each individual production rule. Using this concept elongation is easily modeled with production rules like $F(l) \rightarrow F(l + 0.01)$ as illustrated in the next example:

\[
\begin{align*}
\omega & : F(0.0) \\
p_1 & : F(l) : l < 0.50 \rightarrow F(l + 0.01) \\
p_2 & : F(l) : l = 0.50 \rightarrow F(l + 0.01)[+F(0.0)]F(0.0)
\end{align*}
\]

More detailed information on the use of parametric L-Systems can be found in James S. Hanan’s dissertation [Hanan 92].

### A.3.7 Open L-Systems

Historically, L-Systems were conceived as closed cybernetic systems, incapable of simulating any form of communication between the modeled plant and its environment. In the first step towards the inclusion of environmental factors, Rozenberg defined table L-Systems, which allow for a change in the set of developmental rules (the production set of the L-System) in response to a change in the environment [Rozenberg 73].

Afterwards, Měch and Prusinkiewicz introduced a variant of L-Systems, called *open L-Systems*, which contain a mechanism to incorporate external data in production rules to augment the functionality of environmentally-sensitive L-Systems [Měch 96]. Special symbols, called *communication terms* or *query symbols*, are introduced in the production rules, which can be interpreted whilst rewriting a sequence, replacing them by symbols with associated parameter values such as lengths and rotation angles. This way, specific alterations can be made to the model to take into account the requirements of the environment.

The parameters associated with an occurrence of the communication symbol can be set by the environment and transferred to the plant model, or set by the plant model and transferred to the environment. The environment and
Figure A.3: After each rewriting step, interaction with the environment is performed. During this interaction, query symbols are updated to contain the external information required for further development. The updated plant model is represented graphically and the next rewriting step is executed. (scheme based on [Mech 96])

the plant model are no longer represented by simple functions, but become active processes that react on each other.

In our implementation, communication terms are defined as symbols with question marks as parameters, indicating the request for an information exchange:

\[ \omega : Q(?) \]
\[ p_1 : Q(p) : p = 1 \to A \]
\[ p_2 : Q(p) : p = 2 \to B \]

A.4 Plant Growth based on L-Systems: a Simplified Pipeline

The growth of each tree starts with an axiom, which is the start sequence of the L-System. In every growth phase the current sequence of modules is first replaced by a new one, according to the rewriting productions, using the appropriate probabilities when several productions meet the requirements, matching the parameters used and taking into account each symbols context.

In a second stage, the communication terms are updated. By applying the earlier mentioned turtle interpretation (Section A.2), but without actually drawing the result, the state (and more specific the position) of the drawing tool can be reached for each symbol in the current sequence where communi-
cation with the environment is requested. At this position in the environment information can be queried about illumination quantity, illumination direction, the presence of obstacles, etc. Accordingly, the parameters of each communication term can be determined and filled in appropriately.

Finally, the model can be drawn based on a graphical interpretation of the resulting string and the earlier steps can be repeated as required, introducing new communication terms where required.

Figure A.3 visualizes the main steps from this pipeline.
The work presented in this dissertation is built upon several years of research. This appendix provides an overview of the publications that reported on this research, and were presented at international scientific conferences or were published in computer graphics related journals.

### B.1 Topic Related Publications

The following list of publications contains work that is part of this dissertation:


**B.2 Off-topic Publications**

The next two publications are not addressed in this report:


Samenvatting (Dutch Summary)

C.1 Introductie

Technologische ontwikkelingen hebben ertoe geleid dat men steeds gemakkelijker geconfronteerd wordt met artificiële data. In vele gevallen gaat het zelfs om volledige, virtuele omgevingen. Bij dergelijke toepassingen worden vaak de belangrijkste aspecten van een scène sterk uitgewerkt en benadert men de rest met behulp van eenvoudigere technieken.

Dit is ook het geval bij virtuele, natuurlijke omgevingen. Doordat wij als mens dagelijks geconfronteerd worden met de natuur, hebben wij ons een concreet beeld kunnen vormen van haar werking (bv. hoe een zonnebloem zich richt tot de zon en hoe bomen bewegen doordat wind op hen inspeelt). Wanneer er vervolgens bij het bekijken van virtuele scènes te veel van onze verwachtingen onbeantwoord blijven, heeft dit een effect op onze perceptie van realisme en komt de inhoud artificieel over.

Het creëren van geloofwaardige, natuurlijke scènes blijft een belangrijk probleem binnen computer graphics onderzoek, niet alleen door de grote complexiteit van de aanwezige objecten, maar ook door de vele kleine details die
nodig zijn voor een geloofwaardige representatie ervan. Simpele, intuitieve en interactieve technieken zijn vereist om het modelleren en animeren van bomen en planten te vergemakkelijken. Zo werd het doel van ons onderzoek het vinden van nieuwe technieken die de modeller/animator hierbij kunnen helpen.

Daarom formuleren we de thesis van dit document aan de hand van de volgende twee stellingen:

*Door het simuleren van de omgevingsgevoelige processen die betrekking hebben op plantengroei, kan men het waargenomen realisme van artificiële planten en bomen verhogen.*

*Voor hun animatie, bieden zowel het aanwenden van bestaand video materiaal als het simuleren van de onderliggende fysische bewegingswetten een sterke basis waarvan nieuwe, geloofwaardige animaties gesynthetiseerd kunnen worden op een eenvoudige, snelle en interactieve manier.*

Onze bijdragen richten zich op het structurele realisme van de plantmodellen en hoe ze op een eenvoudige en geloofwaardige manier geanimeerd kunnen worden. Hun fotorealistische weergave, een ander aspect dat bijdraagt tot perceptueel realisme, wordt echter niet behandeld.

Betreffende dit structurele realisme onderzoeken we o.a. enkele fototropismen. Dit zijn fenomenen die betrekking hebben op enkele omgevingsgevoelige eigenschappen van planten, meer specifiek hun lichtgevoeligheid. Voor het animatiegedeelte dragen we bij tot de creatie van plausibele, eindeloze animaties van bomen en planten, gebaseerd op bestaand video materiaal of fysisch gesignaleerde bewegingen.

C.2 Deel I: Plant Modeling

C.2.1 Realisme in de Vorm van Planten

In dit van deze thesis richten we ons op het realistisch modelleren van omgevingsgevoelige plantmodellen. Historisch gezien is er reeds veel werk geleverd binnen computer graphics onderzoek om objecten exact te definieren. De complexiteit van de objecten is vaak voldoende laag om dit te doen aan de hand van eenvoudige benaderingen. Bij natuurlijke objecten is dit echter niet vanzelfsprekend. Een natuurlijke scène kan namelijk bestaan uit honderden bomen, duizenden planten en zelfs miljoenen grassprieten, waarbij elk object opnieuw
bestaat uit talloze componenten. De vraag naar geautomatiseerde methoden om deze complexe objecten te modelleren is dan ook vanzelfsprekend.

In onderzoek gerelateerd aan het onze, kunnen we twee grote strekkingen waarnemen. Enerzijds zijn er \textit{local-to-global} methoden, waarbij de gebruiker kenmerken van de onderdelen van een plant defineert waarna een algoritme deze componenten integreert tot een volledige structuur. Een typisch voorbeeld hiervan zijn de L-Systeem gebaseerde technieken van Prusinkiewicz et al. \cite{Prusinkiewicz90, Prusinkiewicz97}. Anderzijds zijn er de \textit{global-to-local} methoden waarbij de gebruiker globale aspecten van de plantenvorm beschrijft. Vervolgens gebruiken algoritmen deze informatie om de verdere details van het model in te vullen. Een voorbeeld hiervan zijn de technieken van Deussen et al., waarbij de gebruiker een boomstructuur van componenten opstelt die de vorm van de planten beschrijft \cite{Deussen97}.

Naast deze objectgeoriënteerde methoden zijn er ook technieken ontwikkeld die vertrekken van puntgebaseerde representaties, volumetrische texturen of die gebaseerd zijn op fotomateriaal. Tevens is er onderzoek gedaan naar de weergave van grotere natuurlijke scènes waarin planten een strijd voeren voor licht, ruimte en voedingsbronnen.

Gedurende de laatste jaren is men zich steeds meer gaan richten op de omgevingsgevoelige kenmerken van planten. Zo kan men in literatuur over de morphogenesis van planten lezen over drie belangrijke interacties tussen planten en hun omgeving:

- \textit{Lichtinteractie}: Licht is één van de belangrijkste energiebronnen voor de plant. Een plant is dan ook steeds op zoek naar zo veel mogelijk licht en tracht dit te bereiken d.m.v. verschillende natuurlijke reacties (bvb. de lengte van takken en de grootte van bladeren aanpassen, enkel bloemen ontwikkelen bij voldoende lichtintensiteit, etc.);

- \textit{Objectinteractie}: De vorm van planten wordt beïnvloed door objecten in zijn omgeving. Sommige planten, zoals klimop, gebruiken zelfs een object om zich aan vast te klampen;

- \textit{Bureninteractie}: De aanwezigheid van naburige planten heeft een significante invloed op de ontwikkeling van een plant. Planten die in elkaars buurt groeien moeten voedingsbronnen, ruimte en het beschikbare licht delen.
In deze thesis beschrijven we twee technieken om de lichtgevoeligheid van planten te simuleren. Tevens tonen we aan hoe we onze technieken kunnen aanwenden om ook object- en bureninteractie in rekening te nemen.

C.2.2 De Competitie voor Licht en Ruimte: een Eenvoudige Methode

Onze eerste bijdrage bestaat uit een algoritme dat de invloed van licht simuleert doorheen alle stadia van de plantenontwikkeling. We doen dit via het traceren van de paden die lichtdeeltjes afleggen, vertrekkende uit een lichtbron. Verder gebruiken we de functionaliteit van L-Systemen om plantengroei te modelleren en breiden we ze uit met een mechanisme om de invallende richting van het licht te bepalen, daar waar groei plaatsvindt op het plantmodel.

Het algoritme start van een open L-Systeem (meer informatie over dit onderwerp vindt men in Appendix A). Dit herschrijfsysteem beschrijft hoe, gegeven 1 van 4 mogelijke groeirichtingen voor elk plantendeel, de structuur van de plant aangepast kan worden. Die informatie haalt het systeem uit de omgeving van de plant aan de hand van communicatiesymbolen. Telkens het L-Systeem de huidige sequentie van symbolen update, worden in een tussenfase de parameters van de communicatiesymbolen aangevuld met nuttige informatie over de omgeving van de plant. Deze parameters beïnvloeden bij de volgende herschrijfstap de verdere ontwikkeling van de plant.

Door het berekenen van de paden die lichtdeeltjes afleggen in de scène en telkens te registreren wanneer ze een plantendeel raken kunnen we een beeld vormen van de belangrijkste invalshoek van het licht en dit voor elk onderdeel van de plant. De invalshoek van zo’n deeltje wordt vergeleken met 1 van 4 mogelijke groeirichtingen en een teller houdt voor elke richting bij hoe vaak er een deeltje toe bijdraagt.

Naast de richting van het invallende licht kunnen we ook testen vanop welk object het deeltje weerklaatste. Wanneer een lichtdeeltje afkomstig was van een ander plantendeel (via het markeren ervan bij de gebeurde reflectie) geven we het een negatieve invloed op de corresponderende teller. Dergelijke deeltjes bevatten namelijk enkel nog fotobiologisch inactieve componenten van het lichtspectrum en geven aan dat reeds een ander plantendeel de energie van het deeltje heeft benut. Zodoende kunnen we eenvoudig bureninteractie simuleren.

Het gebruik van de open parameters laat tevens toe om te testen of een object aanwezig is nabij een groeilokatie op de plant, zodat ook geanticipeerd kan worden op de nabijheid van mogelijke obstakels. Het gebruik van een
octree laat toe om op een ruimtelijk niet uniforme wijze de dichtheid van takken te controleren en te begrenzen tot een specifiek maximum.

C.2.3 Plantengroei Gebaseerd op een Schatting van de Lichtdichteit

De eerder besproken technieken zijn erg heuristisch van aard en de juistheid van de techniek is erg afhankelijk van de hoeveelheid lichtdeeltjes die gebruikt worden. Onze tweede methode lost dit op door te werken met lichtdensiteiten i.p.v. kwantiteiten en verfijnt enkele technieken om bepaalde natuurlijke fenomenen te simuleren.

Opnieuw traceren we de paden van een verzameling virtuele lichtdeeltjes. Ditmaal slaan we de paden echter op in een octree. Op elk niveau van deze octree bewaren we een referentie naar het pad van een deeltje dat er passeert en we verfijnen de octree enkel rond punten waar informatie over de omgeving opgevraagd kan worden. Op die manier kunnen we, dankzij een goed gekozen maximale diepte voor de octree, ervoor zorgen dat we voor elk positie in de scène een lokaal overzicht kunnen bekomen van de lichtstralen die er voorbij komen. Uit deze beperkte verzameling van stralen kunnen we de lichtdensiteit afleiden en bepalen uit welke richting het meeste licht invalt.

De posities van de communicatiesymbolen in het op een L-Systeem gebaseerde plantmodel bepalen de verfijning van de octree zodat enkel op die locaties in de scene de situatie van het omgevingslicht geschat moet worden. Via de open parameters kan vervolgens de verdere ontwikkeling van het plantmodel rekening houden met de berekende informatie. Na elke groeistap passen we dynamisch de octree aan om de nieuwe geometrie in rekening te brengen.

De geintroduceerde techniek laat toe om verschillende fototropismen te simuleren. Zo kunnen we eenvoudig takken laten buigen, de grootte, positie en oriëntatie van bladeren aanpassen aan de beschikbare lichthoeveelheid, de ontwikkeling van bloemen versnellen of uitstellen, enz.

C.3 Deel II: Plant Animation

C.3.1 Videotexturen van Plantenbewegingen

In eerste instantie focussen we ons op het maken van plausibele, eindeloze videofragmenten van planten uit korte inputfragmenten. Geïnspireerd door de video textures techniek van Schödl et al. [Schödl 00] presenteren we een
eenvoudige methode die het verloop van een gegeven videosequentie aanpast om tot nieuwe fragmenten te komen die een gelijkwaardige inhoud vertonen maar eindeloos kunnen doorgaan of herhaald kunnen worden. Onze methode baseert zich op de observatie uit de natuur dat bewegingen van bomen en planten een erg symmetrisch karakter vertonen.

Eerst presenteren we een metriek die voor elke frame bepaalt hoe symmetrisch de beweging is die op dat moment getoond wordt. Deze metriek is gebaseerd op het vergelijken van de inhoud van frames die op gelijke afstanden liggen voor en na de testframe. Wanneer een frame gevonden wordt in een voldoende symmetrische context, kunnen we deze positie in de videosequentie gebruiken om de beweging om te draaien zonder dat de toeschouwer dit kan waarnemen. Een stochastische waarde overeenkomstig met het symmetrische gehalte van de framepositie geeft voor elke frame aan hoe vaak deze gebruikt kan worden als keerpunt.

Vervolgens beschrijven we een eenvoudige methode om nieuwe videofragmenten af te leiden uit de input data, gebruik makend van de beste symmetrische posities. Een adaptief algoritme zorgt ervoor dat we het beschikbare beeldmateriaal optimaal kunnen gebruiken door het herhaaldelijk omkeren van de video op eenzelfde punt af te straffen en een eerder wisselend gedrag aan te moedigen. Dit kan eenvoudig bekomen worden door de eerder toegekende kansen overeenkomstig aan te passen.

De resulterende techniek laat toe om korte videofragmenten van geïsoleerde planten uit te breiden tot hun eindeloze versies. Daar de bewegingen van bomen turbulenter zijn en hun bewegingen enkel lokaal een symmetrisch karakter vertonen is deze techniek echter niet geschikt voor hun animatie.

C.3.2 Cartoon Animaties Gebaseerd op het Video Textuur

Vervolgens trachten we het repetitieve karakter van de eerder vermelde videotexturen te vertalen naar animaties gebaseerd op getekende keyframes. Via een iteratief proces, kunnen we met behulp van computer geassisteerde keyframe interpolaties deze keyframes uitbreiden tot langere en zelfs eindeloze cartoonanimaties.

In eerste instantie levert een animator enkele vectorgebaseerde keyframes. Via een iteratieve procedure laten we toe om deze keyframes uit te breiden met hun tussenliggende afbeeldingen. Het bepalen van het aantal frames tussen elk
paar keyframes gebeurt aan de hand van een eenvoudige metriek die de posities van controlepunten in rekening neemt. Deze metriek zorgt er tevens voor dat onoverbrugbare transities vermeden worden. Het effectieve interpoleren gebeurt aan de hand van vectorgebaseerde in-betweening methoden.

Het geheel van keyframes en de tussenliggende frames die hen verbinden kunnen we representeren m.b.v. een graaf. Op deze graaf kunnen vervolgens random paden afgelegd worden die de eenvoudige animaties tussen de keyframes koppelen tot een groter geheel. Door het toekennen van een kost aan de overgangen binnen de graaf kunnen we voorzien in een interactieve methode die, gegeven een ‘doel’ keyframe, een kortste pad kiest om dit doel te bereiken. Op die manier kan een interagerende animator de animatie sturen naar zijn verwachtingen.

Dezelfde techniek kan ook toegepast worden op keyframes gebaseerd op fotomateriaal. Van beide methoden worden enkele voorbeelden gegeven.

C.3.3 Boomanimaties Afgeleid van Fysisch Gebaseerde Simulaties

Ten slotte onderzoeken we de mogelijkheid om complexere stukturen zoals bomen te animeren door opnieuw een graaf op te stellen die verschillende poses van de geometrie met elkaar verbindt m.b.v. goedgekozen, interpolerende transities.

Omdat het opstellen van coherente keyposities van bomen een erg moeilijk taak is om manueel uit te voeren, vertrekken we van een fysisch gebaseerde simulatie. Door een mass-spring systeem te koppelen aan de geometrie van een boommodel (dat we bijvoorbeeld bekomen m.b.v. de technieken uit het eerste deel van deze thesis) en er random windkrachten op los te laten kunnen we voor de controlepunten van het model positionele samples opslaan van de verschillende poses die de boomstructuur aanneemt tijdens de simulatie. Voor elk van deze punten vormt de aaneensluiting van hun samples een gericht pad waarmee we de originele bewegingen kunnen herconstrueren.

We gaan echter een stap verder, en breiden dit pad uit tot een gerichte graaf zodat, net zoals bij de cartoonanimaties, met behulp van doelgerichte paden nieuwe bewegingspatronen kunnen gegenereerd worden volgens de wensen van de animator. De nieuwe animaties reflecteren dan de eerder geregistreerde en dus fysisch correcte bewegingen van de boom.

Een methode gebaseerd op het detecteren van kruisende padsegmenten in de originele sequentie, samen met een techniek om transisties te voorzien om
mogelijke afbuigingen vloeiend af te handelen, laten toe om de bewegingen van één gesimuleerde boom eenvoudig uit te breiden met nieuwe data. Na optimalisatie, kan de resulterende graaf gebruikt worden om meerdere instanties van eenzelfde boommodel plausibel te animeren op een niet rekenintensieve wijze.

C.4 Algemeen Besluit

Deze thesis behandelde verschillende aspecten uit het huidige computer graphics onderzoek. Zowel modeleer- als animeertechnieken kwamen aan bod, waarbij we steeds de realistische representatie van natuurlijke objecten zoals bomen en planten in het achterhoofd hielden.

In eerste instantie richtten we ons op het waarheidsgetrouw modelleren van bomen en planten. Door hun omgevingsgevoeligheid, en in het bijzonder hun gevoeligheid voor licht mee in tekening te nemen, zijn we erin geslaagd om plantmodellen te genereren die duidelijk een invloed van hun omgeving weerspiegelen.

Het tweede deel van de thesis focuste zich op het animeerproces. Via inzichten opgedaan bij het opstellen van videotexturen van planten zijn we gekomen tot de creatie van eindeloze cartoonanimaties uit eenvoudige vectortekeningen en fotomateriaal, met behulp van computer geassisteerde interpolatietechnieken. Deze methode hebben we vervolgens uitgebreid naar een 3D variant, die met behulp van een geoptimaliseerde graaf van fysisch gebaseerde boombewegingen van slechts enkele modellen, in staat was om plausibele animaties te genereren van een grotere hoeveelheid boominstanties.


[Coconu 06] Liviu Coconu, Oliver Deussen & Hans-Christian Hege. "Real-time Pen-and-ink Illustration of Landscapes." In Proceedings of the 4th international symposium on Non-


