The Application of L-systems and Developmental Models to Computer Art, Animation, and Music Synthesis

Jon McCormack, B.Sc.(Hons), Grad Dip. Art (Film & Television)

A Thesis Submitted for the Degree of Doctor of Philosophy.

**School of Computer Science and Software Engineering,
Monash University, Clayton, Australia.**

December 2003.

Contents

List of Figures

List of Tables

Abstract

This thesis addresses the development of Lindenmayer systems (L-systems) and associated developmental models for the purposes of creating generative art, animation, and music. The research presented takes formalisms initially developed for the modelling and visualisation of biological systems and extends them as a creative tool for the generation of time-based visual and sonic structures.

New techniques for the geometric interpretation of produced strings are introduced. These techniques improve over previous methods in their flexibility and scope in modelling organic form. Extensions introduced include: the use of generalised cylinders as a biologically inspired modelling tool, a variety of stochastic basis functions, and communication functions that connect the developing grammar with external events and environments. New developments are illustrated by applying them to modelling phyllotaxis, surfaces representing landscapes, and population distribution models over such surfaces. An analytical method for modelling phyllotaxis over arbitrary surfaces of revolution is presented. Temporal and developmental extensions specific to the generation of animated visual models and musical events are discussed in relation to L-systems. A number of related extensions are combined to form a generalised developmental model suited for hierarchical specification of both morphogenetic geometric models and musical events. Examples illustrate the application to the animation of articulated legged figures and the growth and development of plant models. Aesthetic evolution techniques for L-systems, and the application of object-oriented methods suited to synchronisation and real-time performance in practical implementations, are also described.

These technical developments are placed in context historically by examining previous attempts at applying formal systems to artistic applications, and the use of botanical and biological visualisation in creative and scientific contexts. Specific topics on realism and mimesis in relation to computer graphics and views of nature and natural systems are considered. In particular, the concept of emergence is discussed in relation to generative art such as that produced by the formal systems detailed in this thesis.

Declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Jon McCormack Date

Acknowledgements

This thesis is based on my work over the last 12 years in creating artworks and animation unique to digital computers. I am grateful to the many people who have directly, or indirectly, supported the development of my work and provided important inspiration and criticism during this time.

Firstly and most importantly, I am indebted to Gary Warner, who many years ago gave me a copy of Chris Langton's paper on *Artificial Life* from the first workshop held in Santa Fe, New Mexico in 1987. By accident or design (I suspect the latter) this turned out to be one of those seminal moments that went on to define my activities for many years to come. While working at the Australian Film Commission in the early 1990s, Gary was instrumental in funding and supporting my interactive animation, *Turbulence*, at a time when such works were not well understood or supported by cultural funding bodies. Gary's support and encouragement for my work over the years has been without question, of critical importance.

I would like to thank Alessio Cavalaro, Michael Hill, Rachel Kent, Jacqueline Ford Morie, and Deanna Morse, who were all prodigious in supporting the earliest exhibitions of *Turbulence* at ACM SIGGRAPH Florida, USA, in 1994 and in Australia in 1995 at the *Filmmaker and Multimedia Conference* and the Ian Potter Gallery at the University of Melbourne. Ross Gibson advocated securing and funding the redevelopment and acquisition of my work for the Australian Centre for the Moving Image, during his time as creative director.

Few people will be more relieved to see this document complete than my erstwhile supervisor, Peter Tischer, who's encouragement and supply of references has been invaluable. Peter also generously proofread and provided many useful comments on the final version of this thesis.

Alan Dorin, my co-conspirator in the Centre for Electronic Media Art (CEMA), Monash University, has kept me occupied with many valuable discussions, useful suggestions, and his continued enthusiasm and support for many of my more esoteric ideas.

I have had numerous interesting discussions, inspiration and advice from my many colleges at Monash University, in particular Lloyd Allison, John Crossley, Kevin Korb, Binh Pham, and Peter Tischer. I would also like to thank Professor Les Goldschlager who had great faith in offering me an academic position despite my lack of professional credentials. Trevor Dix, David Abramson and Bala Srinivasan have all supported my career and professional development during my time as a university employee.

I am grateful for inspiration, discussions, and information over the years from Mark Bedau, Char Davies, Jim Hanan, Przemek Prusinkiewicz, Bill Seaman, and Karl Sims. In particular, my debt to the inspirational research and ideas of Professor Prusinkiewicz and his colleagues will be apparent to the reader. I would also like to thank Phil Husbands and his colleagues at COGS, University of Sussex, for hosting my position as visiting researcher in 2001, where components of the research for this thesis were carried out.

Sections of this work were made possible through project grants from the Australian Film Commission and funding provided by a New Media Arts fellowship from the Australia Council for the Arts. I am grateful to these organizations for their faith in me and support of my work, even when some of the time it didn't appear that what I was doing was making art (I wasn't).

I would also like to acknowledge the continuing support of Wavefront Technologies Inc., (now Alias|Wavefront Inc.), and Pixar Animation Studios who have generously supported my work by providing me with their rendering software as part of their artist support and university donation programs.

Finally to my family: my wife Julie and my children Imogen and Sophie who have put up with all of this for many years, always with their love and support. I now hope to make up some of the time we have lost. It is to them that I dedicate this thesis.

August 2003. Melbourne, Australia.

1 Prolegomenon[1]

Expelled from the smaller, friendlier world in which previous centuries of men moved with confidence born of familiarity, we are today compelled to cope with an expanded scale of events in a big, alien, redefined world. In order to live freely and fully in our new world, we have to learn to map its strange vistas, to discern in them harmonious structures appreciable by our visual sensibilities, and to arrange our lives in conformity with the new perspectives.

So far, we have failed to live up to the twentieth-century challenge. Science has opened immense new vistas to us, but we have failed to utilize our new technology fully or share it wisely. We shrink from accepting the deeper and richer sense of life, uniquely in our twentieth-century world, that is sometimes touched upon in the best moments of our best artists. We have not yet found our places in this broadened world.

— *Gyorgy Kepes (Kepes 1944)*

The state of knowledge at the beginning of the twenty-first century suggests the complex nexus of relationships between art, science, and nature are, paradoxically, becoming increasingly detailed and accessible, yet simultaneously more complex and unfathomable. Our experiential understanding of what constitutes nature and the natural is blurred, and with an eye on the progress of science and technology ensues a shifting redefinition of the way we view the world: experientially, epistemologically, ontologically and culturally.

In recent decades, the computer has had a profound impact on our knowledge and understanding of the world. Computer modelling and simulation

[1] Meaning "introduction" — used in the tradition of the late Alain Fournier, whose articles were inspirational for the origins of the research presented here.

have opened up immense new vistas, but in the spirit of Gyorgy Kepes, it is poignant to question if we have used our new knowledge and technology to create a deeper and richer sense of life. This thesis attempts to address that question. As the philosopher John Gray reminds us, while science has enabled humans to satisfy their needs, it has done nothing to change them (Gray 2002). If we accept that knowledge and theories cannot only be considered in isolation, we are then required to address a broader range of issues in relation to that knowledge and those theories. How concepts are applied and engaged with: the purposes for which they are used and the modes under which they are explored, mandates consideration as well if we are to address Kepes' challenge. Such exploration feeds back into the knowledge understanding and discovery process, and therefore has a role in determining *what* is discovered and *why*.

The research presented in this thesis takes formalisms initially developed for the modelling and visualisation of biological systems and extends them as a creative tool for the generation of time-based visual and sonic structures. The rational and context for this design is investigated within the theoretic and historic framework of applying formal systems to creative applications. These systems are broadly classified as *generative systems*, whereby some formal process is enacted to generate new systems, properties, and structures. These features are distinguished by a far greater novelty and complexity over that of the system specifying them. *Generative design* involves the implementation and orchestration of autonomous processes by the designer. *Generative art* involves the use of such generative systems and processes for creative purposes.

The core of the generative system described here is general enough to have scientific and artistic value. The results and representations used are mainly confined to the development of an aesthetic theory, which is the primary motivation for this research.

1.1 Motivation

The research described in this thesis began as an exploration of ideas in botanical modelling from a computer graphics perspective. Foley et. al. regard computer graphics as "a branch of computer science … but its appeal reaches far beyond that relatively specialized field" (Foley et al. 1994, page 1). Research in computer graphics finds motivation from, and application in, a diverse set of fields which include: scientific visualisation; simulation; engineering; marketing; product and prototype design; feature films and television; medical imaging; office automation; art. Not all techniques are applicable to all application areas. Computer graphics, as I use the term in this thesis, I define to mean the development of mathematical and computational models with the goal of a *perceptually coherent visual representation*. That is, in the broadest sense, they produce patterns that can be interpreted by the viewer. The results of my research and the practical experimentation that ensues from it were developed with the goal of making dynamic and interactive creative works, both visual and musical. As this process developed, I began to think more about the relation between computer graphics and art, and to search for theories and discourses in art to place my own work in context with elements of art theory (in addition to its utility in computer graphics per se). This led to a deeper analysis of the relationship between computer graphics and art, and inexorably science and art as a whole. As a result, this thesis includes discussion and analysis not typically found in a work that deals primarily with a branch of computer science. It is my view that science and science as a cultural activity are not easily separated. As Gyorgy Kepes challenges: science may open up the vistas, but it is up to individuals to develop "the deeper and richer sense of life".

So while science may have some justification to its claims of objectivity in the abstract, the application of scientific theories, their realisation as technology, even the justification and selection of which particular areas are researched does not appear to be "value-free". As Hansen's slogan reminds us: "seeing is a theory laden enterprise" (Hansen 1958). This is particularly the case for computer graphics, where much research is *applied science*: applied to the process of visualising the world. In computer graphics, the goal of most research is the simulation of "reality", be it a computer scientists, engineers, movie-producers, botanists, or even photographers view of reality. Even when computer graphics deals with the non-real, it still tries to emulate a particular approach, in a way that is beheld to be perceptually faithful (real), not to the style or feature itself, but to primarily a screen-based version of that style or feature. Everything real is ultimately expressed through the fundamental quanta of pixels. This is true too of artistic styles[2] where the success of a simulation is in how real it appears in relation to the analogue version of the technique, as viewed through the limitations of the CRT or other graphics display technology.

Is it possible to use computers in general (and computer graphics specifically) to create artworks that are not based on the standard focus of realism, or designed to borrow visual styles and methodologies directly from other media? Certainly, this is one of the goals of the work described here.

Aside from any issues of correctness, thoroughness and readability by which any scientific paper would be judged, much research in computer graphics is implicitly judged on how "real" or true the images or animations are to the element or aspect of reality they try to reproduce. However, if the subject of the simulation has no analogue in reality, this form of validation breaks down. In addition, as will be detailed in the following chapters, if our goal in being real is in explaining some held truth about a representation, the literal visual version of this truth may not be the best way of imparting it.

If we are interested in developing scientific and technical processes for artistic purposes, it makes sense to look at theories of art that might be useful in understanding how art theory views the world, and in particular the concepts of "copying nature" and representing nature in art.

The primary motivation, however, is in extending and applying the class of mathematical formalisms, known as *Lindenmayer systems* (L-systems) to applications in computer graphics, fine art, and music synthesis. The concept of *generative processes*, of which L-systems form a part, has a rich and diverse history in many aspects of creative exploration. A significant motivation for the use of generative processes is that the results they produce are often *emergent*. I use this term in the sense that emergent processes can "exceed the designers expectations" in terms of the behaviour, form, pattern or interaction that they produce. Hence, this thesis illustrates the results of an investigation into using L-systems as a basis for modelling complex, emergent developmental phenomena, suitable for creative applications.

[2] Often referred to as "non-photorealistic rendering" in the computer graphics literature.

1.2 Original Contributions of this Thesis

1.2.1 Background

The main body of research and development on which this thesis draws began in 1991. At this time, I embarked on the creation of an interactive artwork that would revolve around the philosophical and technical mediation of nature and natural history. Much of the research for this work involved developing a software system capable of visual simulation of natural systems. To this end, I used L-systems as the basis of my work.

Over a number of years, I developed a variety of extensions, algorithms and methods related to the goal of an experimental system capable of automating the tedious production of computer animation by capitalising on the generative power of the digital computer. The interactive artwork, titled *Turbulence: an interactive museum of unnatural history* was first exhibited in 1994. In order to produce this work, I had developed a number of technical advances related to the automated generation of animation using L-systems. These advances were driven by the practical consideration of using the system to actually create and produce an artwork that exploited the generative features found in biological systems.

Following this production, the system was further extended to the generation of musical structures in addition to visual ones. In an attempt to unify a number of different models based on L-systems, reaction-diffusion systems and cellular models, I devised the cellular developmental model, described in detail in Chapter 9.

1.2.2 Research Context

This research draws from a wide variety of sources, including art theory, the history and philosophy of science, critical theory[3], computer graphics, visualisation, biological modelling, and computer music. Due to this broad base of

[3] Critical theory traditionally is inspired by the German philosophers of the "Frankfurt School", such as Theodor Adorno, Max Horkheimer and Herbert Marcuse. Critical theory takes society itself as an object of study and attempts to explain how knowledge emerges in human sciences and society. Critical theory distinguishes itself from traditional scientific enquiry in that critical theory is always mindful of the entanglement between theory and the circumstances in which it is developed, whereas scientific theory has what Horkheimer terms "instrumental reason" in which objects are external to the theory describing them (Horkheimer 1937).

enquiry, few readers could be expected to be expert in all these areas, so an important first step is to survey key research and ideas from these areas, as they relate to the original research presented in this thesis. My contributions can be summarised in two principle areas:

- An analysis and discourse investigating artificial life ("A-life") art and artists in representing nature as aesthetic object and as generative system. This analysis is defined in terms of an art theory that describes and influences the production of generative artistic works.
- Research into the concept of *emergence,* and how emergence relates to generative systems, particularly those concerned with novelty and creativity.
- The development of time-based, hierarchical, developmental formalisms, based on L-systems, which find application in the synthesis of three-dimensional form and the creation of musical patterns. The goal of this research is in producing new kinds of creative visual and musical works that autonomously change and develop over time.

To quote art theorist and critic Jack Burnham, I am "seeking sustenance in the austere soil of the scientific model and technical invention" (Burnham 1968a).

Due to the long period over which the research described here has been undertaken, a number of other researches have independently described similar or related techniques to those described here. For example, the use of generalised cylinders to model complex shapes in plant modelling detailed in Chapter 6 of this thesis has not been previously published even though it was developed for the production of *Turbulence* in 1994. An analytic method for generating phyllotaxis over surfaces of revolution, and the generation of large-scale landscape models illustrate how these geometric modelling advances can be applied, in addition to their own utility in using L-systems to model natural form.

My use of artificial evolution of L-systems dates back to 1993 (McCormack 1993) and was the first time such techniques had been applied to L-systems for this purpose. An updated version of this publication can be found in Chapter 10.

The application of L-systems to music composition (discussed in Chapter 11), is based on material I first published in 1996 (McCormack 1996). This chapter describes a number of new methods for generating music from L-systems, including the use of hierarchies to mirror compositional structure and the use of polyphony and context to create musical sequences of greater musical complexity than has been previously demonstrated by other researchers (detailed in Chapter 11).

The developmental cellular model, introduced in Chapter 9, unifies a number of different L-system formalisms and discusses methods for real-time generation and synchronisation. Such methods are crucial when the model is used for interactive music generation or real-time interactive graphics applications, for example.

1.3 Related Work

Related work is divided into three broad (and sometimes overlapping) categories. I distinguish technical methods primarily developed by artists for expressing a creative idea or work, which I have loosely labelled *"Artificial Life Art Systems"*; these are detailed in Section 3.6.6. I would include my own work in this category. This I differentiate from related technical work with a different emphasis, for example contributing to theoretical computing and language formalisms, botanical modelling, or simulation. This area I have called *"Generative Modelling Systems"* (Section 1.3.1), though a significant proportion is related to the use of Lindenmayer systems (L-systems) in plant modelling. I use this more general term since much of the contribution of this thesis is in extending techniques pioneered by L-systems to a creative domain. Hence, some related systems have not necessarily focused on L-system formalisms to achieve their goals as generative modelling systems. Finally, there have been a number of critical and art-based studies, which analyse generative and A-life methodologies for creative applications. These are summarised in Section 1.3.2.

1.3.1 Generative Modelling Systems

Much of the work in this thesis related to L-systems and developmental botanical models is based on and influenced by the pioneering work of Przemyslaw Prusinkiewicz and his colleagues at the University of Calgary over the last 16 years. Building on the theoretical model developed by Lindenmayer (Lindenmayer 1968), Prusinkiewicz applied the turtle interpretation of produced strings to generate realistic three-dimensional plant models (Prusinkiewicz 1986b), further developed in (Prusinkiewicz, Lindenmayer & Hanan 1988). A number of developments followed, including parametric extensions (Hanan 1992; Prusinkiewicz & Hanan 1989); animation techniques (Prusinkiewicz, Hammel & Mjolsness 1993); synthetic topiary (Prusinkiewicz, James & Mech 1994); environmental interaction and open L-systems (Mech & Prusinkiewicz 1996); visually realistic and complex ecosystem models (Deussen et al. 1998) and interactive plant modelling system that incorporates positional information from user-edited parametric curves (Prusinkiewicz et al. 2001).

The book, *The Algorithmic Beauty of Plants* (Prusinkiewicz & Lindenmayer 1990) summarises the important contributions of the authors in the application of L-systems to plant modelling and computer graphics research into the modelling of natural phenomena (at the time of its publication — many important developments have been made since then, e.g. see (Prusinkiewicz et al. 1995)). *The Algorithmic Beauty of Plants* served as the original inspiration for my involvement in research into L-systems and their application to generative art.

The L-system modelling programs, `cpfg` (Prusinkiewicz, Hanan & Mech 2000) and `L-studio/cpfg` (Prusinkiewicz et al. 2000) have some degree of intersection with the L-system modelling programs developed in this thesis, although they are primarily directed at generating realistic visual models of plants.

Similarly, the *xfrog* modelling program developed by Lintermann and Deussen is an interactive plant modelling system which uses a visual programming technique to generate plant models for computer graphics (Lintermann & Deussen 1996, 1998, 1999). The *xfrog* system places an emphasis on giving the user expressive graphical interaction for interactive design, at the expense of offering the generality and flexibility that hand coding of complex L-system productions permit.

Shape grammars (Stiny 1975) are also related to the work described here. The pioneering work of George Stiny and James Gips was one of the first attempts to develop an aesthetic theory based around generative computer models which used Chomsky-like grammars to build and design aesthetic objects (Stiny & Gips 1978). Shape grammars specify a series of shape rewriting rules and operate directly on geometry, rather than via an intermediate symbol representation as is common with L-systems.

1.3.2 Critical and Art Theories

Attempting to define a role for art in relation to science is not new. In a certain sense, art has had a continuing relation to science since the origins of science itself. Some of this history is detailed in Chapter 2.

In modern times, the work of Jack Burnham stands out as an important precursor to current positivist engagements of art and science. Burnham's book, *Beyond Modern Sculpture*, pre-empted the focus of A-life art today. Burnham and his contemporaries are discussed in Chapter 3.

In terms of locating the related fields of A-life art, generative art and new media art, Mitchell Whitelaw has studied the work of a number of A-life artists, including my own work (Whitelaw 2000). He sees particular historical relevance to the work of the artists Paul Klee and Kasimir Malevich in the diverse creative practice of a number of A-life artists (Whitelaw 1999). Stephen Wilson provides an overview of A-life art in his book *Information Arts*, and surveys a number of prominent artists working in this area (Wilson, S. 2002).

Critical engagements with a number of artificial life researchers have been carried out in the humanities. N. Katherine Hayles looked at the meta-narratives developed in artificial life, in particular the work of Tom Ray and his *Tierra* program. She comments on the narratives being developed in artificial life research, and how they are used in order to fit the agenda of researchers who seek to redefine the definition of "life" (Hayles 1996).

Anthropologist, Lars Risan studied artificial life researchers and concluded that for some approaches to artificial life, it was a "technoscience leaving modernity" (Risan 1997). Risan comments on the methodological closeness or distance between subjects (the researchers) and objects (the machines that are studied), seeing influences and similarities in twentieth century continental phenomenology, rooted in the philosophy of Martin Heidegger.

Stefan Helmreich, also an anthropologist, spent time at the Santa Fe Institute to develop his theory of a "second nature" created by artificial life researchers that "mirrors the first nature" (Helmreich 2000). Helmreich also looked at gender roles and sexual stereotyping in a-life simulations, however a number of his assumptions have been criticised, even from within his own profession (Hayles 2001).

1.4 Overview of the Thesis

The remainder of this thesis is divided into two parts. The first part looks at the theoretical and critical background associated with using science and biological processes to produce *generative* artworks. The second part of this thesis details the technical research based on L-systems and similar developmental models to realise generative temporal and sonic structures.

Chapter 2 develops an introductory analysis of the basic systems used in art and science and also covers some specific attempts of science to study and explain art and artistic behaviours. Chapter 3 examines the concepts of beauty and realism, or *mimesis* and its relation to computer graphics and "naturalism". Realism is a central concept in computer graphics so it is poignant to examine this concept in relation to art theory if computer graphics techniques are to be used in an artistic setting. The work of a number of photographers who used botanical imagery as their principle subject are also examined as a precursor to the images generated using the techniques described in part 2.

Chapter 3 also introduces concepts regarding *generative art,* looking at historical and contemporary examples, both computer and non-computer based. It is argued that generative processes place particular emphasis on the concepts of *emergence* and *hierarchies* — the subject of Chapter 4. From these discussions, a theory of the *computational sublime* is developed, examining similarities and differences with historical theories of the sublime. The concepts and theories developed in this chapter serve as a guiding philosophy in determining the motivations and design decisions for the technical research presented in the following chapters.

Part 2 describes the technical component of the research. Chapter 5 gives an overview of L-systems as formalisms used for generative modelling, including deterministic, stochastic, context sensitive, context free and parametric L-systems. Chapter 6 describes the use of *turtle interpretation* of strings, introducing techniques that extend the range and complexity of model generation over previous methods. Key enhancements include the use of generalised cylinders and the addition of stochastic and other functions to control development of models. These enhancements are applied to a number of examples in Chapter 7, which also introduces a new method for generating phyllotaxis over surfaces of revolution. Techniques discussed in the previous chapters are applied to the modelling of landscapes and vegetation, as an illustration of the practical application for the techniques developed.

Chapter 8 deals specifically with animation and extends previous uses of timed L-systems to applications in computer animation, such as continuous animation of morphogenesis and growth, and the animation of legged gaits in animals. A description of the practical algorithm used to simulate timed L-systems is also detailed.

A number of previously described variants on L-systems are unified in the cellular developmental model of Chapter 9. This model represents a developmental system that combines discrete and continuous techniques in a hierarchical specification. Novel contextual and interpretative schemes allow application of the model to generation of both music and geometric form. The design of an object-oriented developmental and synchronisation system gives the system the ability to generate data in real-time and to maintain synchronisation with external timing sources. This is particularly useful in applications involving human-computer interaction and real-time musical performance.

Chapter 10 extends the design of L-system grammars to include an evolutionary component whereby the grammar evolves according to the aesthetic criteria of the operator. A review of major developments in this area since the original publication (on which the chapter is based) was written, are also provided in an epilogue.

Chapter 11 focuses on the application of L-systems to music generation, describing how grammars can be devised that open new possibilities for composers, as well as being capable of simulating a number of other popular music generation techniques such as Markov models and Petri-nets.

Finally, Chapter 12 summarises the thesis and provides some conclusions.

Part 1

2 Art and Science

Language sets everyone the same traps; it is an immense network of wrong turnings. And so we watch one man after another walking down the same paths and we know in advance where he will branch off, where walk straight on without the side turning, etc. etc. What I have to do then is erect signposts at all the junctions where there are wrong turnings so as to help people past the danger points.

—*Ludwig Wittgenstein. (Wittgenstein & Wright 1980)*

Art and Science are two very broad terms that encompass a wide range of activities, ideas, theories, and opinions. Most observations that can be made at this "top level" are restricted to being crude generalisations. Yet, it is at this level that many discussions and much of the literature takes its focus. One might observe that, for every example proposing some generalisation about relationships between art and science, a counter-example can usually be found. Nevertheless, in any effort to detail work of a cross-disciplinary nature, entry is often easiest via the synoptic overview.

This chapter looks at the basic premises of art and science. Some of the important philosophies and methods of each are detailed. As this is a broad overview, observations will be general and it is difficult to cover in detail what amounts to a major portion of human intellectual and creative activity. Nevertheless, some of the key differences will be examined with the desire to frame and situate discussions in the following chapters. In addition, some important efforts by science to "explain" artistic and creative behaviours are examined.

An important component of this thesis is to place in context the primary motivations and design decisions that have influenced the original research presented in later chapters. In many cases, such motivations and design decisions are established as part of the cultural presuppositions of the particular field of enquiry.

In the case of the work described in this thesis, the aim is not only to describe the relevance of the work to the fields of computer graphics and computer music, but also to place in theoretical and historical context the application of technical components to the artwork they are designed to create. While the methodology and manner of presentation of this thesis follows a scientific format, the motivations and goals have significant artistic components. Hence, the research presented here draws from both artistic and scientific disciplines.

2.1 Language and Methodology

Science and art have evolved to become two quite distinct fields. For most of modern history, they have blissfully coexisted predominantly with indifference or ignorance to each other. There have of course, been notable exceptions, some of which will be examined later in this chapter.

Given the multi-disciplinary nature of this material, this section will look broadly at the methodologies and philosophies of art and science and introduce some essential concepts and disciplinary-specific terminology. However, unlike cross-disciplinary studies in the sciences, the so-called "bridge laws" (Nagel 1961) that span scientific disciplines are not so easily constructed. Understanding contemporary art and science is not as simple as detailing definitions or explaining jargon. Many of the terms used by science are also used by art: "creativity", "knowledge", "discovery", "genius", "real", "beauty", "understanding", for example. But an analysis of how these terms are used and what they attempt to describe reveals that, like the iceberg, there are many layers of implicit theories, constructions, epistemologies, and even ontologies hidden underneath. Fundamentally, art theories and science theories are largely built on different philosophical systems. These systems, for the most part, exist as separate enterprises, suited to developing the agendas of the disciplines that are constructed on top of them.

In addition to their predominantly distinct philosophies and agendas, art and science differ radically in their approach. The physical sciences, for example, cumulatively develop theories, hypotheses or laws based on empirical experiment, models, simulations and logical propositions with a view to explaining and understanding the natural world in a rational and coherent way. Discussions about art are based around *critiques* and *discourses* — a term generally understood to "comprise an organized body or corpus of statements and utterances governed by rules and conventions of which the user is largely unconscious" (Macey 2000, page 100). This is not to suggest that all art critiques and discourses are irrational and incoherent (though this may be so for some), on the contrary, many discourses in the arts have provided valuable insights beyond their original frame of reference.

It could be argued that science pays less day-to-day critical attention to its philosophical and cultural assumptions than art, because it has been so successful in the "progress" of modern society. In political terms, science's practical results and its power of prediction are often seen as justification enough for its methods, without the need to directly defend or qualify the methodology and assumptions of science itself. The latter half of the twentieth century has seen the formal establishment of the history and philosophy of science, which seeks to explicitly examine and understand science's claim to being a special kind of knowledge. This period has also seen the development of studies in the humanities and social sciences, also with an interest in studying science, but from a different perspective. Some sectors of the humanities, seeing science's success and power, have tried to adopt some of science's methodologies and practices, with a view that being more scientistic will make them more powerful voices of knowledge.

2.2 Science and its Philosophies

2.2.1 The Origins of Science

Science has its origins with the ancient Greeks. Wolpert argues that, unlike forms of creative practice, which are ubiquitous in human culture, science as a discipline originated in human culture only once, in ancient Greece[4] (Wolpert 1993). In the sixth century BC, the Milesian philosophers were looking for the *archê* — the origin or principle of all things. They asked the question: "what is reality made of?" Thales of Miletos (624–546 BC) theorised that the world was made of water, because of its multiple states, ubiquity on earth, and its essential use for life. Anaximenes (585–528 BC) thought air, Anaximander (610–546 BC) thought reality made from "the boundless" from which everything is created and eventually returns. A number of authors see some essential characteristics from the philosophy of Thales[5] and his Greek contemporaries that forms the basis of Western science: an emphasis on rationality; a mode of thought that distrusted everyday perceptions of the world and "common-sense" beliefs; and, as opposed to myths and supernatural explanations, theory that was self-consistent and open to argument, debate, and critical analysis.

2.2.2 The Enlightenment and Beyond

The new science of the seventeenth century brought about major publications and discoveries in science, its nature, and method. With the Enlightenment came the first modern encyclopaedias and its philosophical contributions pushed the envelope in trying to liberate humanity through rational thinking. This "age of reason" is typified by Kant's motto "*Sapere aude:* have courage to use your own understanding" and Goya's famous etching from the *Los Caprichos* series showing a philosopher fallen asleep at his writings, surrounded by nightmarish creatures with the caption reading: "The sleep of reason produces monsters".

In the centuries that followed science was increasingly associated with progress, spurring unparalleled changes to Western society. In the late nineteenth and early twentieth centuries, major contributions to mathematics,

[4] Wolpert goes to great lengths to differentiate science from technology, which he argues has originated in many different cultures.

[5] Some debate exists over exactly what can be attributed to Thales, since no written works by him survive. His philosophy regarding water is attributed to Aristotle in his *Metaphysics*. Describing Thales observation that "all things are water", Brumbaugh, writes that this "...may seem an unpromising beginning for science and philosophy as we know them today; but, against the background of mythology from which it arose, it was revolutionary." (Brumbaugh 1981)

logic, physics and biology “made for an unprecedented excitement in science” (Machamer 2002, page 1).

2.2.3 Logical Positivism

As the twentieth century progressed, logical positivism became an important philosophical movement in relation to the validation of scientific laws. The logical positivists saw the factual basis and empirical grounding of science as superior to other knowledge systems. In support of this view, they attempted to formulate and solve the problem of the nature of empirical meaning. This problem was solved in two parts. Firstly, using first-order predicate logic to formally specify sentences in scientific theories, and secondly, relating theory to observation via the *verification* principle. This principle was used to connect theory with experiment and observation.

The positivist view of scientific methodology was static and universal, based in empiricism and logic. Their goals eventually proved untenable: by the 1940s logical positivism had given way to a number of other doctrines, notably logical empiricism. Machamer notes that the positivists were “the first to see the problems with their program” and that “virtually all the major moves that were to come later and so change the character of the philosophy of science were first initiated by the positivists themselves” (Machamer 2002).

2.2.4 Contemporary Historical and Social Studies of Science

With the establishment of formal studies in history and philosophy of science, emphasis shifted from the positivist agenda of modelling scientific theories as axiomatic systems, to looking at the history and process of science. In *The Structure of Scientific Revolutions*, Thomas Kuhn divides scientific history into periods of “normal science” in which the dominant conceptual framework (which he terms “paradigm”) prevails (Kuhn 1996). These periods of normal science are punctuated by “paradigm shifts”, where new paradigms come into play due to the increasing untenability of the existing ones. An oft-quoted example is the “revolution” in mechanics from the theories of Newton to those of Einstein. In these shifts, there is *meaning variance,* where the meanings of previously axiomatic properties change (for example, the property of “mass” has different meanings in Newtonian and Relativistic mechanics). Old paradigms die out because eventually they cannot answer the questions they raise. Kuhn argues that there is no rational basis for the change from one paradigm to another, and thus revolutions must be explained in terms of social processes. Differing versions of “theory loading” in science have also been developed by Feyerabend (Feyerabend 1985) and Hansen (Hansen 1958). A number of exceptions have been shown to Kuhn’s basic hypothesis, for example in the recognition that a scientific theory contradicts observation only being recognised after the theory has been replaced by a generally accepted new theory (Wolpert 1993, page 103).

Nonetheless, if one looks at science from the point of view of history, history shows that in numerous cases, what was once held to be commonly accepted scientific truth — has turned out to be wrong.[6] Belief and knowledge systems are always in a state of flux, and based on this assumption, there is no reason to believe that today the same argument does not apply. There may be new “paradigm shifts” that await science and dispel what is commonly held to be “scientific truth”. In a sense, however, one of science’s strengths is that it can anticipate such shifts, because it is a cumulative knowledge system. Paradigm shifts do not necessarily render the old paradigm obsolete. Newtonian physics is still taught to students of physics, and for the majority of engineering applications it still is the dominant paradigm, due to its predictive successes. This suggests that scientific theories can be validated in other terms, such as their utility.

Interpretations with a focus on the history and process of science have lead to a general thinking of relativism in scientific theories. If social processes determine the acceptability of competing theories, then to some extent, paradigms or theories are determined by social constructions. Facts are constructed rather than discovered.

2.2.4.1 The Demarcation Problem

Another important characterisation of science is often called the *demarcation problem*. One of the most popular answers to this problem comes from Karl Popper. Popper used as his starting point Hume’s induction problem — the

[6] This was essentially the view of Nietzsche, who argued that the values by which we judge the reliability of knowledge are subject to historical change, so can we be sure we have the right values. See (Faith 2000).

inferring of relationships from repeated instances as being logically untenable. In the case of scientific statements, Popper argued, induction was a key element used to support their universality. For example, we cannot infer that because we have seen only numerous white swans that *all* swans are white. Popper proposed the solution of *falsifiability*, whereby scientific statements are not confirmed by positive verification, but are subject to potential falsification by succeeding and more accurate statements. Popper saw scientific theories as “nets which attempt to ‘catch’ the world, to explain, rationalise and master it” (Magee 1982; Popper 1968).

According to Worrall, it is now (almost) universally accepted that Popper’s account fails (Worrall 2002). Warrall also points to the work of Duhem and “auxiliary assumptions” as a difficulty with the falsification proposal that existed even before it was proposed by Popper (Duhem 1906). There are of course, many examples where falsification of scientific theories has been ignored with positive results[7], so the criterion of falsifiability does not completely encompass the scientific methodology.

2.2.5 Realism and Anti-Realism

The issue of falsifiability raises another reoccurring issue with science, in its perceived claims to objectivity and truth — claims that give science its elevated status, social and epistemological currency. The validity of such claims rests on differing philosophical viewpoints, broadly classified as *realist* and *anti-realist*. Scientific realism asserts that the entities described by scientific theories exist and that the theories themselves are objectively true (and independent of the representation used). Theories opposed to realism include positivism, empiricism, instrumentalism and constructivism (Wilson, R. A. & Keil 1999, page 707). Realist philosophies have been influenced both from scientific discoveries (e.g. quantum theory (Fine 1986a, 1986b) and from arguments within philosophy itself, for example the “miracles” argument (Putman 1975; Smart 1963). In the latter part of the twentieth century, realism has come under increasing debate, further fragmented and revised from previous definitions. For example, *entity realism* limits itself to the inde-

[7] See for example, Boyles repeated attempts to prove the existence of a vacuum in (Shapin, Hobbes & Schaffer 1985)

pendent existence of theoretical entities and not the truth or otherwise of the theories that employ them.

More recent alternatives to realism, that still encompass some of its original propositions (in spirit at least), include: Putman's "internal realism", which allows scientific claims to be "true from certain perspectives, but denying that science tells the whole story, or even that there is a whole story to tell" (Fine 1999); Van Fraassen's "constructive empiricism", which forgoes truth for empirical adequacy (Van Fraassen 1980); and Fine's "natural ontological attitude", which "regards truth as basic but, seeing science as open, it challenges general prescriptions for scientific truth, including the perspectivalism built into internal realism and the external-world correspondence built into realism itself" (Fine 1986b, 1999).

Constructivism, on the other hand, maintains that knowledge is socially constructed — that "facts" are made rather than existing. This view has been used to counter science's claims to truth and objectivity (as do many others), and leverages a general relativism to the weight of scientific beliefs.

2.2.6 Science as Social Activity

Paul Feyerabend saw the power of science as social rather than epistemological and claimed that when analysed sufficiently, scientific propositions were as meaningless as those of metaphysics, and critiques the rationality of science as a whole (Feyerabend 1975). For Feyerabend, evidence and proof are seen as rhetorical, and theories proffered by those with the most power and rhetoric becomes accepted.

Pessimistic accounts of science's claims to objectivity and truth, such as those advanced by Kuhn and Feyerabend were themselves influenced by the more general ordinary language philosophies associated with Austin and Wittgenstein. In the social studies of science, science is viewed as a social activity, and like all social activity should be studied in the socio-cultural milieu wherein that activity takes place (Latour 1979). This includes the rules of social engagement and social and cultural dynamics by which scientists develop power structures over each other and society in general.

2.2.7 Science, Rationality, Modernity

In general terms, modernity is the period that signifies a break from Classical Antiquity, beginning with the Enlightenment. In criticisms of modernity, science is seen as a key player in the definition of the modernist, post-Enlightenment worldview, where individualism and personal freedom is achieved from a foundation of scientific knowledge and rational values. This focus on rationality is seen as the basis for social and personal progress, made possible by material labour and self-controlled work.

Critics see modernity as "a movement of ethnic and class domination, European imperialism, anthropocentrism, the destruction of nature, the dissolution of community and tradition, the rise of alienation, [and] the death of individuality in bureaucracy" (Cahoone 1996). A rationalist view does not seem to fully account for what it is to be human, in the sense that the pure rationalist doctrine seeks to "eschew feeling, emotion and other distortions to knowledge in favour of pure reason" (Coyne 1999, page 5). Nor has it been able to achieve its desires for the liberation of society by virtue of values founded on reason alone. As echoed by the failure of logical positivism, rationalism has thus far failed to provide a complete understanding of the objectivity of knowledge. If one sees limits and brittleness in human knowledge then a rationalist viewpoint cannot offer the certainty it seeks to provide (nor can any other).

2.2.8 Postmodern Narratives

In the view of postmodern philosophers such as Lyotard, science forms one of many *grand narratives* or *metanarratives* — narratives that make forms of knowledge legitimate by supplying them with a validating philosophy of history (Lyotard 1984; Lyotard & Benjamin 1989). Metanarratives are seen as universal with the ability to explain all other narratives by translating them into each particular metanarrative's language. Lyotard defines postmodernism as incredulity towards metanarratives, where the grand narratives of modernity have collapsed, to be replaced by "little narratives" none of which has genuine claims to specialised status or power.

Postmodernism is a blanket term describing both the period "after modernism" and a collection of vaguely related movements in philosophy, architecture, and the arts. In popular (mis)understandings, it has been synonymous with the notion that all forms of knowledge, taste and style are relative, and the political, logical, even scientific arguments used to justify superiority of one over another are invalid. However, many key theorists and critics reject the reading of simplistic relativisms so popular in perceptions of postmodern theory (Clark 2002, page 4).

The postmodern rejection of absolutism has not been confined to the humanities or arts. Even in mathematics — a branch of science with perhaps the strongest claims to objectivity and truth — a number of authors suggest that mathematics has several ontological difficulties (Chaitin 1999), that is much more empirical than previously thought (Chaitin 2002; Tymoczko 1986) and might even be considered “postmodern” (Tasi´c 2001).

2.2.9 Discussion

Arguments over metaphysics, realist and anti-realist philosophies, and the social construction of science have had only minor effects on “the scientific methodology[8]”, in the sense that such philosophical arguments are not normally part of the day-to-day concerns of scientists, or limited and applicable to science alone. In addition, science’s predictive utility and explanatory power continues to reinforce its status and legitimacy. In the arts, however, the objections and incongruity of some philosophies to science have been more widely exploited. These will be more fully detailed in the next section.

2.2.10 Summary

The goal of this discussion is not argue for any particular point of view, either in favour of, or against, science and its claims. Scientists may hold different realist or even relativist philosophies and still be able to practice science successfully. A testament to science’s power as an epistemic system is in its ability to cope with radical changes and paradigm shifts. The purpose here is to survey some of the major contributions as far as they are influential in the development of contemporary views of science from outside science itself. In

[8] Few would argue that there is really such a thing as *the* scientific methodology — in contemporary scientific practice, scientists may pursue any number of vastly different methodologies in investigation and research (Wolpert 1993).

the main, the arts draw their intellectual and philosophical ideas from elements of the "human sciences"[9], art theory, cultural studies, and the somewhat ironically named area of "critical theory" (Macey 2000, pp. 74-76). In simplistic, but fundamental terms, these disciplines do not see science in the same way that science (or even the history or philosophy of science in the analytic tradition) sees itself. When trying to cross borders between art and science, there will inevitably be conflicts and disjunctions, often glibly summarised with a reference to the "two cultures" of C.P. Snow[10] (Snow 1959).

It would show some naïveté to claim to make absolutely no value-judgements about science as described here. In writing this thesis, I have, in general, adopted an approach based on a conventional scientific methodology and its written conventions. This must implicitly indicate at least, the value and respect that I judge science to have.

However, when it comes to discussions about artwork and artistic practice, such methodologies and conventions are not always appropriate, and in these cases discussion has been framed in the context of an arts discourse. My hope is that such distinctions will be clear to the reader in the pages that follow.

2.3 Art and its Philosophies

Today, art has many meanings. The Encarta dictionary definition includes reference to "the creation of beautiful or thought-provoking works, for example, in painting, music, or writing" and "beautiful or thought-provoking works produced through creative activity". Such definitions seem to fail to capture the broad range of activities carried out as "art". Unlike science, art, or at least some form of creative behaviour, is universal to human cultures (Brown, D. E. 1991). My goal here is not to define art in general, or even cover all its philosophies. Science has a much more definite and definable philosophical heritage, whereas art might be summarised as a history of rejection and change. Unlike many authors, I see little that can broadly characterise art in any meaningful way, rather I see greater benefit in focusing on particular

[9] Generically defined to include the fields of anthropology, history, psychology, sociology, and linguistics.

[10] Although Snow's concerns were based more around a fashionable disciplinary ignorance rather than contradictory systems of knowledge and belief (Cordle 1999).

vectors of art that I deem most relevant to those associated with the notions of art as expressed later in this thesis.

2.3.1 Philosophical Traditions

Much of the history and philosophy of science as discussed in the previous section comes from the Anglo-American tradition, often referred to as *analytic philosophy*. In general, analytic philosophy, its discourses and traditions, are distinguished from *continental philosophy* — a covering term for modes of thought that originated in Europe. Analytic philosophy has had little interest in European philosophy since Kant. Hence, the two traditions have evolved with scepticism, ignorance and indifference towards each other. From a scientific perspective, much continental philosophy can appear impenetrable, vague, nonsensical, even wrong[11], which may be why literature, cultural studies and artists more often embrace it.

Science expects explanations to be rational and coherent — the hallmark of any good scientific theory or hypothesis is that anyone, given enough time and resources, could learn it at any time[12], without recourse to "mysterious incalculable forces", poetic, or multi-layered readings. Art, because it does not necessarily adhere to the modernist rationalist doctrine often embraces irrationality and relativism just as enthusiastically as the more "traditional" concepts of rationality and beauty, for example.

2.3.2 Definitions of Art

The practice and domain of art has undergone radical changes in western society over the last two hundred years. These changes reflect political, structural, social, and intellectual changes to life and culture, many fuelled by science and technology. According to Wilson:

> Previously, art was produced in historically validated media, presented in a limited set of contexts for a circumscribed set of purposes, such as the search for beauty, religious glorification, or the representation of persons and places. … this century has generated an orgy of experi-

[11] As illustrated by Alan Sokal's famous "spoof" paper published in *Social Text*, which liberally quoted continental philosophy, and the follow-up article exposing the fraud (Sokal 1996b, 1996a).

[12] This was the sociologist, Max Weber's thinking on the meaning of science. (Weber & Eisenstadt 1968)

> mentation and testing of boundaries. New technological forms, such as photography and cinema, have already raised questions about art. Artists have added new media, new contexts and new purposes. (Wilson, S. 2002)

Spurred by many dialogues in philosophy of art, critical theory, and cultural studies — particularly those involving epistemological and cultural relativisms, deconstructive analyses and post-modern discourses — art's definition, its meanings, status, and role in society have diffused and fragmented to the point where "anything goes"[13], and almost any object, activity, or idea can be called "art". Artist Robert Irwin has said that art "has come to mean so many things that it doesn't mean anything any more".

In a similar vein, Wilson quotes the Getty Museum Program in Art Education:

> Art-making may be described at the process of responding to observations, ideas, feelings, and other experiences by creating works of art through the skilful, thoughtful, and imaginative application of tools and techniques to various media. The artistic objects that result are the products of encounters between artists and their intentions, their concepts and attitudes, their cultural and social circumstances, and the materials and media in which they choose to work. (Wilson, S. 2002, page 17)

A self-referential definition such as this gives little insight into the process or qualities of modern art, and is perhaps, conspicuous by its supposition that art must involve resultant objects.[14] Interestingly, one could substitute the word "scientist" for "artist", "scientific" for "artistic", and "Science" for "Art-making", and still get a reasonable result.[15] Attempts to give all-encompassing definitions fail to capture what is unique about specific types of art or an artistic practice. Many individual artists seek definitions that aid in bringing a sense of legitimacy to their particular practice, as this cannot be addressed by all-embracing definitions. For example, Irwin seeks to define art's role as "a continuous examination of our perceptual awareness and a continuous expansion of our awareness of the world around us".

[13] Or perhaps more accurately, "anything you can get away with".

[14] In the political sense that the Getty museum improves its status through its collection of unique objects, so it is in its interest to define "art-making" (rather than art) as producing objects, since it cannot so readily collect more intangible things like performance or conceptual art.

[15] One could also substitute almost any profession that involves producing things through interaction with the world, which highlights the paucity of this definition.

There is no right and wrong or even a relative scale in between when discussing ways of being an artist or of making art. This is a kind of category error, which seeks a semantic solution to a different problem, one that cannot be framed in such a context. However, this does not mean that art rejects critical analysis or that it cannot be examined by theory.

2.3.3 Art Theory and Art Criticism

In discussing issues of art we can make a broad distinction between *art theory* as developing discourses and ideas around collections of artworks, artistic ideas or artistic practices, and *art criticism* as developing critical opinions and dialogues around particular art movements, artworks or artists. Freeland describes the difference between scientific theory and art theory thus:

> Art theory is not like scientific theory in that it's use for prediction or general explanation is minimal. There does not seem to be any laws of art that will predict artists' behaviours, or that explain the 'evolution' of art history by detailing what 'succeeds' in making a work beautiful or significant. (Freeland 2001)

2.3.4 The Beautiful

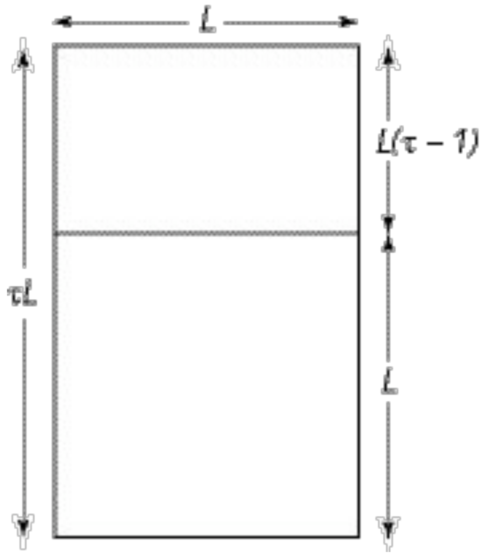
> They [scientists] tend to expect artists to be interested in beauty, and to be surprised to find that we have a stock response that beautiful things must be sentimental or facile.
>
> —*A.S. Byatt (Ede 2000, page 10)*

Throughout most of recorded history, theories of aesthetics and beauty are a consistent subject when discussing art. For example, Plato, within his philosophy of ideals (such as form) and universals saw the arts as a kind of craft, with artistic practice as a form of *mimesis* or imitation (covered in more detail in the next chapter). Aristotle however defended imitation, particularly in epic tragedy and poetry, as an instinct and source of both knowledge and pleasure. In the thirteenth century, Thomas Aquinas developed a theory that beauty was an essential or "transcendental" property of God. Hence, artworks should seek to emulate and aspire to God's awe-inspiring properties, reflected in the formalist traditions of *proportion*, *light*, and *allegory* as used by medieval cathedral builders. The transcendental nature of art is still important today, even though it has shed its religious origins.

Edmond Bourke's differentiation of the beautiful from the *picturesque* and *sublime* is indicative of a change in art theory in the eighteenth century, particularly towards landscape (the sublime will be covered in more detail in Chapter 4). Following Bourke, Kant developed an extensive treatise on the concepts of beauty and the sublime.[16] He tried to explain how we make judgements about the universality of an artwork's "beauty", deciding that true beauty was a property of the object itself rather than a subjective judgement of the observer. For Kant, art was judged by taste, nature by beauty, and the concept of nature's subjective "purposiveness" *(Zweckmäßigkeit)* being judged aesthetically.

2.3.4.1 Proportion and Beauty from Nature

A continuing theme in art, architecture and design is the use of special proportions, such as the golden ratio[17] (Figure 2-1) as a basis for beauty in the designed form.



$$\tau = \frac{1}{\tau - 1} \qquad \tau = \frac{1}{2}\left(1 + \sqrt{5}\right) \approx 1.618034$$

Figure 2-1: The golden ratio (τ). A golden rectangle (left) with sides in the ratio 1:τ can be partitioned into a square and new rectangle. This new rectangle has sides with the ratio 1:τ.

[16] Kant's treatment of Beauty was far more developed than Bourke, and it was not only framed in relation to the Sublime.

[17] Also referred to as the "golden section" or "golden mean".

This particular ratio has been documented in the architecture of Stonehenge (12th–16th centuries, B.C.), the ancient Greek architecture of the fifth century, B.C., through the Renaissance and on to the present day. Leonardo Da Vinci was fascinated by branching structures, finding that the cross-sectional area of daughter branches summed to that of the parent. Studies by Fechner in the late nineteenth century, and later by Lalo in 1908 show that the majority of people surveyed have a natural preference for rectangles proportioned to the golden ratio (Elam 2001, pages 6-7).

The golden ratio is also found in a variety of natural objects, including the phyllotaxis observed in flower heads and pinecones (the subject of Section 7.1); the spiral growth of seashells and the body proportions of animals (as studied by D'Arcy Thompson, for example (Thompson 1942)).

A detailed study of proportional harmonies in nature, art and architecture was undertaken by the architect György Doczi, who showed special harmonic relationships in natural and artificial structures, with particular emphasis on the golden ratio (Doczi 1981). Doczi also showed how musical systems exhibit similar harmonic relationships, thus establishing an aesthetic connection between natural form and musical scales. He invents the term *dinergy*[18] because while many terms "refer to aspects of the pattern-forming process of the union of opposites…none expresses its *generative* power". Doczi's key emphasis was that in nature harmonies are dynamic, generative processes that result from the union of opposites: this union forming a harmonious proportion. For example, his "leaf study" reveals the harmonic order of vein structure in a lilac leaf, showing that proportional relationships approximate the golden section and the pentagonal Pythagorean 3-4-5 triangle (Figure 2-2). These find correspondence in the diapente (fifth) and diatessaron (fourth) musical harmonies, giving rise to an aesthetic connection between nature and music (Doczi 1981, page 11).

[18] Made from two Greek words: *dia*— "across, through, opposite;" and "energy".

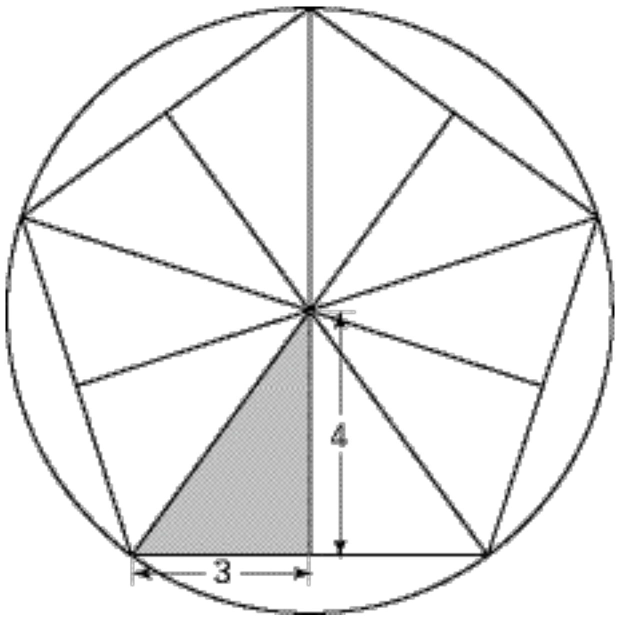


Figure 2-2: The 3-4-5 pentagonal triangle from (Doczi 1981).

I carried out my own leaf study (Figure 2-3), which shows comparable results to that of Doczi (with ratios approximating the inverse golden ratio and a 1:1 ratio for the leaf chosen).

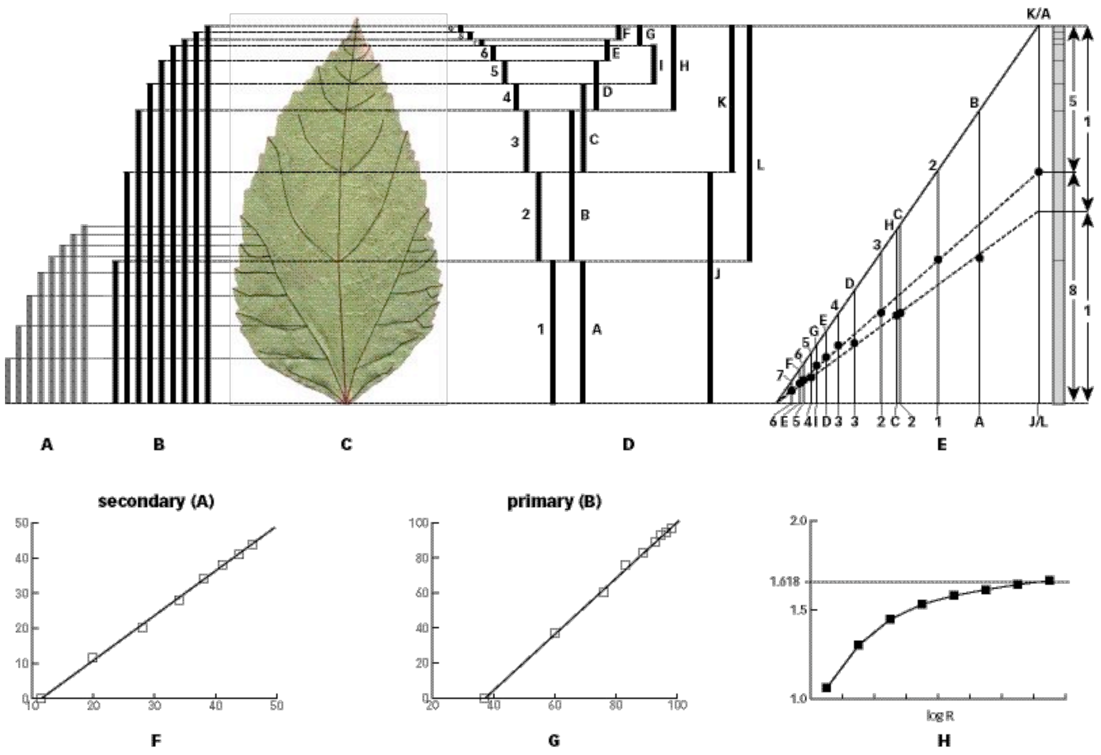


Figure 2-3: 'Leaf Study' (after Doczi) carried out by the author shows the analysis of harmonious proportions of a leaf. Distances of veins branching from the midrib (**C**) are shown as heights (**B**), secondary branches are also plotted (**A**). Distances are numbered from 1—9 (**D**). A number of neighbouring distances are grouped and labelled A-L. The distances

and their groupings are aligned in pairs and plotted (**E**). Two approximate ratios emerge, one approximating the inverse golden ratio (0.618) and the other an approximate 1:1 ratio between successive vein groupings. Plots of alternating distances show approximately constant ratios between branching distances, both in the primary (**G**) and secondary (**F**). The log plot of branching ratios for secondary veins approaches a limit of 1.618 (**H**).

As shown in the figure the main vein structure of the leaf (**C**) is analysed. Distances between the starting points of the veins along the midrib are measured and form harmonic orderings. In the case of Figure 2-3, both the primary veins from the midrib and a selection of secondary veins branching off the first primary vein are measured (**B** and **A** respectively). The distances between successive veins are calculated for the primary veins and numbered from 1 to 9 (**D**). Other groupings of distances, created by accumulating neighbouring distance measurements, are also calculated. These groupings are labelled A-L. These distance measures are lined up by neighbouring pairs, (**E**), which show the two main ratios between measurements. Pairs of distances form points in two-dimensions and can be plotted in sequence. For n distance measurements, the pairs $(0,1),(1,2)\ldots(i,i+1),(i+1,i+2)\ldots(n-1,n)$ are plotted. The plots **F** and **G** are obtained by this method for the secondary and primary distances respectively. Finally, a log plot of the ratio between successive branch distances (points at which vein branching occurs) approaches a limit that approximates the golden ratio.

Similar to the leaf study, relationships are described by Robert Bringhurst showing harmonic relations from music and nature as a basis for page layout and typographic structure for the printed page (Bringhurst 1992). Such relationships date back to medieval times and the earliest days of the design of the printed page.

2.3.4.2 The Loss of Formalisms

Thinking about art as no longer purely a matter of form (and form's loss of previously held spiritual and mysterious powers), lead to a general abandonment of formalism in art. From the late nineteenth century, classical concepts of beauty and direct imitation became less and less important to many art movements, given the increasing plurality of ideas in western art, liberated by rapid changes in culture and technology. Modernist searches for a universal formal quality of beauty have continued into the twentieth century, suggesting that “beauty” as an idealised concept may have survived as a topic of continued interest. But the twentieth century also brought about many diasporic hybrids and so more recently, post-modern, post-colonial, feminist, and post-structuralist theories deal with the concept of beauty in ways that bear little resemblance to classical or modernist concepts.

Today, Western art opens and admits many more possibilities than in any previous time. As one recent overview admits that art “includes not just works of formal beauty to be enjoyed by people with ‘taste’, or works with beauty and uplifting moral messages, but also works that are ugly and disturbing, with a shatteringly negative moral content.” (Freeland 2001)

Or, as critic Matthew Collings summarises it:

> ‘Why do we have to conceptualise anything’ is one of the main anxieties that society has about the art of now. Within art there are many anxieties too because art expresses the anxieties of the society among other things. But it rarely expresses the main one directly, the anxiety that art is vacuous now. (Collings 2000, page 225)

2.3.5 Summary

In summary, the purpose of this section has been to acknowledge that art, as it is understood today, does not only mean or include the classical, traditional or even modernist realisations or ideas, found in the fine arts of, for example, painting and sculpture. As reflected by changes in life and culture, art may be instantiated in a variety of new and traditional media; be ugly, disturbing, or morally repugnant; it may even be conceptually vacant (Francblin & Baudrillard 1991). Such factors, however, should not preclude developing a consistent theory or scholarly discourse around a contemporary art practice. What is important is for the individual to define meanings and definitions within their own particular practice.

2.4 Science on Art

Science has made various attempts to study art, in the sense of accounting for artistic behaviour in humans from psychological, behavioural, evolutionary, and neuroscience perspectives. In addition, research has focused on areas related to art, such as perception and creativity. The basis of most scientific analysis of art and artistic behaviour resolves to reduce art to its universal roots and propose a materialist explanation for those roots.

One of the earliest modern attempts to formalise creativity was by the mathematician G. D. Birkhoff, who endeavoured to define the aesthetic measure of objects to be the ratio of their symmetry to complexity (Birkhoff 1933). While moderately successful for simple shapes such as polygons and some vases, the theory broke down when attempts were made to generalise the measure to a broader range of art objects.

Humphrey sees humans drawn to beauty as a dog is drawn to saccharine — there is an innate desire to find beauty in certain constructs of *likeness tempered with difference* (Humphrey 1973). Humphrey sees aesthetics as a biological predisposition of humans and animals to seek classification of structure in the world around them. Beautiful structures facilitate classification since they provide evidence of possible taxonomies in ways that are easy to understand.

2.4.1 Neural Mechanisms of Artistic Experience

A study by Ramachandran and Hirstein proposed "a theory of artistic experience and the neural mechanisms that mediate it" in the Journal of Consciousness Studies (Ramachandran & Hirstein 1999). Their experiments suggest that visual art provides "super-stimuli" that excite certain regions in the brain more strongly than natural stimuli, and that artists consciously or unconsciously develop rules or principles to "titillate these visual areas of the brain". Like the evolutionary psychologists, Ramachandran and Hirstein seek to find universals that define creative behaviour and response in humans. Their thesis is in part based around the *peek-shift principle* from animal learning: that learning is not based on prototypes, but rules. Artworks are in some sense caricatures of prototypes, with the exaggeration or difference from the prototype form, resulting in a form with relevant features that are more prominent than in the prototype form itself.

Responses to this study from an artistic perspective, criticised the article's narrow understanding of art (referring to it as "our propensity to create and enjoy paintings and sculpture"), the patriarchal language used, and the ignorance of the influence of culture on interpretation of aesthetics and beauty by such reductionist and materialist analysis (Mitter 1999). Such responses are typical to the science/art divide — that when one discipline studies another each make implicit assumptions or simplifications about language, definitions and knowledge systems, or, even more unfortunately, they fail to gain sufficient understanding of the opposing area under study, resulting in simplistic or inappropriate generalisations. Nonetheless, the idea of developing a neural understanding of certain types of creative behaviour and response seems to hold some potential.

2.4.2 Biological Basis of Creative Behaviour

In disciplines such as evolutionary psychology, investigators look for the biological origins and the psychological relations of art. Investigating human creative behaviour from a biobehavioral and ethological viewpoint, anthropologist Ellen Dissanayake suggested that the biological could offer better explanations than the metaphysical in this regard. She identified three major features that suggest art as a biological adaptation. Firstly, that art making in some form is a universal human behaviour (Brown, D. E. 1991). Secondly, it is costly, often involving considerable time and effort. Thirdly, that it is pleasurable to both creator and viewer (Dissanayake 1988, 1995).

Art-making, in the sense of creating aesthetic objects with some cost to the organism, but with no apparent direct survival advantage, is not unique to humans. Many animals practice or can learn some form of creative behaviour (Diamond 1992, Chapter 9), and it is suggested, as with animals, creative behaviour has a biological basis. Most accounts begin by showing that art's apparent lack of biological utility is only an illusion, then illustrating how art functions in terms of improving mating chances. According to Pinker, the arts engage "not only in a psychology of aesthetics, but also a psychology of status" (Pinker 1997, Chapter 8). Pinker, drawing on work by Wolfe (Wolfe 1975) and Bell (Bell, Q. 1976) argues that the *value* of art is largely unrelated to aesthetics (something that many postmodern theorists would agree with) but confers status to both its creator and those who possess it. Such status is only maintained if the value of the work is conferred socially.

In his book, *The Mating Mind*, Geoffrey Miller states there are two strategies science can take in order to understand the evolutionary origins of art

— focusing on the high-arts (the “Western modernist arts infrastructure”), or the “bottom up” approach looking at the everyday aesthetic productions and experiences of a diverse range of human groups (Miller, G. F. 2000, Chapter 8).[19] Like Diamond, Miller uses the Bowerbird as an example of another species that exhibits creative behaviour. Miller postulates that art may have evolved through sexual selection, offering various possibilities that account for creative behaviour — runaway sexual selection, fitness indication and sensory biases (extended phenotype effects, as proposed in (Dawkins 1982)). Most authors who discuss the biological impetus of art also point out that “modern” or post-modern art is a complex nexus of social and cultural systems, beliefs and ideas and that the biological accounts tend to focus on general “creative” behaviour, not specific to a particular mode, medium or practice. Nonetheless, their views differ as to the ultimate significance of biology to contemporary art theory. Diamond (Diamond 1992) states:

> Thus, human art has come far beyond its original functions. But let us not forget that even the greatest art may still serve those primal functions.

Whereas, Dissanayake suggests that the two types of understanding are different and should not interfere with each other. However, she concludes by suggesting that contemporary art is in conflict with its origins: “what the arts were for, an embodiment and reinforcement of socially shared significances, is what we crave and are perishing for today” (Dissanayake 1988).

Accepting basic abilities and desires for art-making as adaptations does not limit the potential for art to offer experiences of significance in an art-gallery context (Nake 1998). Owning a Porsche to increase ones status through conspicuous consumption does not deny that a Porsche is a clever piece of engineering.[20] It remains, however, disappointing that the majority of present-day contemporary art courses do not include biology as part of their syllabus and that theories of the biological origins of art are treated with scepticism in the arts.

[19] Miller seems unaware that the “everyday aesthetic productions and experiences of a diverse range of human groups” have been the subject of investigation and production in the “high-arts” as well.

[20] This is not to ignore the obvious political implications regarding wealth and status, and the ownership of “great works of art” to a privileged few.

2.4.3 Perception

Many scientific and psychological studies of perception have been influential on art. For example, the Gestalt psychology (Ellis 1938; Koffka 1935) initially concerned itself with perceptual organization in the relationships between parts and wholes, resulting in a kind of *perceptual holism*. Important aspects of Gestalt perception include grouping principles (Wertheimer 1938), the figure-ground relation (Rubin 1921), and frames of reference. Recent studies in perceptual development suggest that principles of organization develop during the first year of life, in contrast with the Gestalt view (Kellman & Spelke 1983).

2.4.4 Discussion

Science has made many varied attempts to study art. From the point of view of science, many of these attempts are seen as credible and successful.[21] In general, from an arts viewpoint, they are treated with suspicion and scepticism. This may be due to, for example, deconstructive or social theories of science that (if accepted) limit science's claim on being a special kind of knowledge.[22] Or, it may be simply that science's understanding of what art is bears little resemblance to the artist's conception of it, or that the activities and behaviours defined as "creative" and "artistic" seem overtly naïve or simplistic and fail to account for the complexity, subtlety and diversity of art.

2.5 Science Relations in Art

Despite many pessimistic accounts of science from some domains, a number of art movements have embraced science or scientific methodologies in some form. Note, that this relationship does not necessarily mean in a way that scientists develop their science or scientific methodologies. In Section 2.2.1, I noted the differentiation between science and technology. Not all authors see this distinction so vividly, for example, Penny refers to it as a continuum with

[21] "Credible and successful" meaning they have been accepted by peer-reviewed journals, and conferences of international standing.

[22] Note that the reciprocal may be true as well — that if art was "explained" or shown not to be a special kind of knowledge, or at least did not have some inexplicable component, then it too would lose much of its cultural mystique and power.

no clear boundary[23] (Penny 1996). Increasingly, as science has progressed and diversified the differentiation between science and technology has become less clear, particularly in a general sense, which inherently these terms themselves are. Therefore, many movements that sought to embrace "science" may have more accurately sought to embrace "science and technology" or even just technology. A more detailed examination of some particular art movements (Section 3.6) will look at this issue more closely.

2.5.1 Art Movements and Science

Futurism utilised the elevation of speed and movement as a celebration of technology and science. Futurism was influential on *Constructivism*, which sort to develop an artistic system out of the characteristics of the real world, in that it sought representation of the fundamentals of nature as proclaimed by science, rather than through perceptual mechanisms or phenomenological means (Fenton 1969).

Hungarian artist, Laslo Moholy-Nagy continued the constructivist tradition, seeing technological change as an accelerator of social progress. His kinetic light sculptures, created in the 1930s, can be considered "generative" machines that created futuristic temporal displays of light in motion. Moholy-Nagy's idea of creating virtual volumes by tracing the paths of objects in motion is seen as a precursor more contemporary digital installations (Paul 2003, page 13).

Many of these movements believed that utopian social changes could be brought about by technological advancement. This advancement could be facilitated or accelerated by art — bridging the gap between our emotional lives and social progress. In art critic Terry Fenton's criticism of Kepes's theories, he sees Kepes as being forced to speak in terms of potential, rather than achievement. This is a continuing theme in associations with art, science, and technology where the utopian progress of digital creativity is always expectant, continually residing in the future (Coyne 1999, Chapter 1).

[23] Penny uses the terms "technology" and what he calls the "engineering world view" somewhat interchangeably.

3 Real / Natural

Art is not a copy of the real world. One of the damn things is enough.

— Attributed to an essay on Virginia Woolf. From (Goodman 1968)

When an object is exactly like another, it is not exactly like it, it is a bit more exact.

— Jean Baudrillard.

The previous chapter surveyed philosophies, relations and studies between art and science in a broad and general sense. In this chapter, I look at three critical issues for developing a visual arts practice that uses computational process as its formal basis. These issues are:

- *The notions of realism and representation in art and in computer graphics.* In order to employ computer graphics techniques in making art, it makes sense to differentiate computer graphics for its own sake (or the sake of the many other applications to which the techniques may be put), and using computer graphics to make art. Realism is one of the major research goals for computer graphics and a significant basis for how it is judged within the field. For art however, realism is a vexed but well-explored concept. Through this discussion, I hope to make it clear why producing "realistic" computer graphics (or even realistic botanical models) is not necessarily the only way to judge results, even though systems described in lattter chapters have been used to create realistic images of plants.
- *Related historical traditions in art.* Rarely do new forms of art come from nowhere — ideas and theories in art seem to resurface in different

forms (Foster, H. 1996), so an examination of related historical art movements is appropriate. In particular, I identify and examine three different areas:

i. *Botanical art and photography,* which situates itself at the border of art and science, and is a precursor to modern scientific visualisation.

ii. *Systems art:* a movement in art that followed on from Constructivism. It's principle ideas can be found in the writings of Jack Burnham and Gyorgy Kepes. It has close relations to cybernetics and cybernetic art, and the systems theory of Von Bertalanffy.

iii. *Romantic and sublime traditions in art* and their newer interpretations in technology based art. This is covered in more detail in the next chapter.

- *Related contemporary work in the areas of A-life and generative art.* The software and artwork described in later chapters can be classified as *generative* art. In this section, I briefly introduce these areas and survey related work.

3.1 Computer Graphics and Realism

One of the major goals of computer graphics is the synthesis of realistic images, often referred to as *photographic realism* or *photorealism* (Foley et al. 1990, Chapter 14). Essentially, the intent is to reproduce certain visual perceptual qualities of reality, in a similar way that a photograph or certain paintings capture "reality". What constitutes realism in a manufactured[24] image has been the subject of much debate (Hagen 1986). A common test in computer graphics to validate the realism of a particular synthesis algorithm is to show test subjects two photographs: one of a real scene, the other of a computer simulation of that scene, and ask each test subject to decide which is the photograph of the real scene, and which the simulation. This suggests, as the name implies, photorealism attempts not to simulate reality, but to

[24] Meaning human-made, e.g. photograph, painting, computer generated image.

simulate photographs of reality[25], as evidenced by the numerous algorithms designed to simulate flaws and artefacts of the photographic process; such as lens flare, depth of focus and motion blur. Foley et. al suggest that if the ultimate goal of a picture is to convey information, then a picture that is free of shadows and reflections may well be more successful than a *tour de force* of photographic realism. They go on to illustrate how simulations of impossible or staged conditions can often make an image *more* "realistic". Hence, even from within the goal of realism in computer graphics, we see that making an image more accurately real can distract from the information the image seeks to convey.

The idea that an image conveys information is crucial in both scientific and artistic applications. In the classic text of Nelson Goodman, *Languages of Art*, he argues that all manufactured images show *symbols* subject to *interpretation* (Goodman 1968). This interpretation is always done within a cultural context, and hence can be subjective or culturally dependent (scientific images often require specialist interpretation based on the scientific culture under which they operate, for example).

3.2 Realism and Photorealism in Art

The concept of painting realistic *"trompe-l'oeil"* (fool-the-eye) images dates back to the ancient Greeks. Later, in Renaissance painting, a mathematical theory of perspective combined with improved painting technologies spurred on the artistic obsession with visual realism. In the eighteenth century, *Naturalism* referred to an artist's fidelity in rendering appearances of the world. Like many art movements, Naturalism found application in painting, sculpture, and literature.

Realist art has its origins in mid-nineteenth century France. It is exemplified by the painter Gustave Courbet, a pioneer of the movement who justified realism by saying that "having never seen an angel, he could certainly never paint one". In this nineteenth century sense, Realism meant not only trying to capture nature in a visually realistic way, but also choosing subject matter

[25] Donald Greenberg in the ACM SIGGRAPH video "The Story of Computer Graphics" states that his research goal was to synthesise images with the computer that were "indistinguishable from real world images" (Foster, F. 1999).

that was real or faithful to the artist's experience. Here, the idea of a realistic image implies a *truth* that extends beyond the image itself. Realism was a reaction to previous traditions in art that focused on the extraordinary depictions of mythology, religious events and the sublime — each in their own way trying to capture what was beyond the artist's experience.

Realism and Naturalism gave way to more interpretative techniques such as impressionism and can be seen in a number of ways as a reaction to the limitations of *trompe-l'oeil* painting in representing reality.

Photorealism is also a term given to an art movement that grew out of the pop art of the 1960s. Principle practitioners included Ralph Goings, Richard Estes, Robert Bechtle, Audrey Flack, and Charles Bell. Many of the practitioners of photorealism literally made "paintings of photographs" (Chase & Goings 1988) — the photographic image serving as a basis for their view of reality.

In their book *Remediation: Understanding New Media*, Bolter and Grusin develop their theory of "Remediation" — the negotiation of one media through other media (Bolter & Grusin 1999). Bolter and Grusin argue that photorealism relies on "the cultural assumption that that the photograph has a special relationship to reality" [p.120], justified by the "automaticity of the photographic process, which draws its images with Talbot's 'pencil of nature'[26]".

> Photo Realism is an art of many ironies — not the least of which is that the artist seeks a directness in relation to the visually experienced world through the use of secondary source material, and that he achieves a heightened sense of reality by reproducing an illusion of an illusion. With his use of the photograph, the artist actually gains a double immediacy.
>
> — Linda Chase, quoted in (Bolter & Grusin 1999, page 121)

A further irony in photorealism is that the justification for simulating reality is because the situation (or reality) being simulated would be impossible with conventional models (Foley et al. 1990, page 607). If a situation is impossible in reality, then a simulation of that impossibility is obviously not a simulation of reality.

From these examples, one could conclude that what *reality* appears to mean, is not that the scene depicted is *real*. Rather, it is that the spatial, textural and illumination cues suggest a correspondence to perceptual cues from

[26] 'Talbot's "pencil of nature"' refers to the book published by photographic pioneer William Henry Fox Talbot (b. 1800 – d. 1877).

the world, when viewed through the mediation of a photograph or pixel-based display medium. These cues are, in general, organizationally and culturally sufficient to suggest a cognitively plausible model of space and object.

While acknowledging these ironies and assumptions, it would be simplistic, however, to consider the goal of photorealism research in computer graphics as singularly the synthesis of images indistinguishable from photographs. Since the earliest days of computer graphics research, an emphasis has been placed on the synthesis of *dynamic* environments. Here the simulation not only includes the behaviour of light/surface and surface/surface interactions, but of developmental and dynamic interactions, such as the simulation of rigid and soft body dynamics (Barzel 1992); objects and entities with amorphous properties (Blinn 1982; Nishimura et al. 1985); or features ill-suited to fixed surface-based representations, such as gasses, clouds, dust, etc. (Reeves 1983). In computer graphics, realism focuses on both the *static* and *dynamic* image. The dynamic visualisation used in graphics is itself an illusion: visual dynamics are most often conveyed as a rapid succession of static images, giving the illusion of animation.

In a general sense, a simulation is considered successful in the context of computer graphics if it demonstrates a likeness to the perceptual qualities associated with the developer's expectations of reality, within the context of the languages of cinema, television and photography. For example, Phong shading (Phong 1975), a technique to simulate specular highlights on surfaces, has no real physical[27] basis in its implementation (Watt & Watt 1993, Chapter 2) — yet it *looks* acceptable enough for it to be used in almost every major commercial computer graphics software system. While refinements in illumination models have been developed that *do* have a basis in physics (Cook, R. L. & Torrance 1992), the perceptual qualities are ultimately the pragmatic factor for acceptance of any technique.

Similarly, particle systems (Reeves 1983), used to simulate amorphous phenomena such as fire, smoke, water and clouds, rely on the idealisation from physics — a particle as an imaginary object with a finite mass yet infinitely small size — a convenient approximation for the purposes of an efficient simulation.

[27] Meaning from the laws of physics or derived via empirical experiment.

The driving force behind these compromises is the trade-off between time and accuracy, understanding and complexity. A model is by necessity a simplification that seeks homomorphic relations between model properties and the phenomena that it seeks to describe. Developing complex animated sequences requires large amounts of computer time and detailed mathematical models. If the ultimate goal is images or animations that appear realistic, then the assumption is that, in general, the viewer is not concerned with the factual accuracy of the simulation, only its perceptual coherence within the tradition and language of photography and realist painting. Thus the homomorphic properties of such models are morphisms to cultural traditions as much as empirical observations.

3.2.1 Perceptually Real and Real

Why is it important to make distinctions between perceptions of reality and reality itself? In many art movements, artists have tried to express their perceptions of reality, with vastly differing results. Within various phenomological philosophies that negate subject/object distinctions (Clark 2002; Dreyfus 1991) this expression can take on many more diverse forms than those expressed by photo-realistic computer graphics.

A dialogue about imitating reality, or *mimesis*[28], has its origins in ancient Greece. In Plato's dialogue *Cratylus,* Socrates describes how Zeuxis painted grapes so realistically that birds tried to eat them. Such an anecdote suggests the paradox that an imitation exists only if we can *perceive* some difference between it and the original (Macey 2000, page 254). If that difference does not exist, then to the observer it is a replica like the grapes painted by Zeuxis. It also highlights the perceptual context — qualities of similarity are confined to those properties provided by the context of the comparison. The birds would soon discover that the grapes were not "real" the moment they tried to eat them.

In a further discussion of the story of Zeuxis and his grapes, Hal Foster retells the story as told by the psychoanalyst Lacan as a "*trompe-l'oeil* contest" between Zeuxis and Parrhasios (Foster, H. 1996, pp. 112-113). Zeuxis paints grapes in a way that entices birds, but the birds are not scared away

[28] From the Greek meaning imitation.

by the image of the boy that holds them (much to the frustration of Zeuxis, for if he had rendered the boy perfectly the birds would have been scared away). Parrhasios, however paints a veil in a way that deceives Zeuxis, who asks to see what lies behind the veil and "concedes the contest in embarrassment". Foster argues that the animal is lured in relation to the painted surface, whereas the human is deceived in relation to *what lies behind.*

> A perfect illusion is not possible, and, even if it were possible it would not answer the question of the real, which always remains, behind and beyond, to lure us. This is so because the real cannot be represented; indeed, it is defined as such, as the negative of the symbolic, a missed encounter, a lost object. (Foster, H. 1996, page 112)

To make distinctions between the real and the perception of the real is to engage one of the most famous of philosophical topics. Experiments in visual psychology demonstrate that perception is not a direct mapping of reality; many artefacts and illusions are easily demonstrated (Anstis 1999), leading to the (flawed) question "how do we know what reality really is if the only way we can experience it is through our senses?"

Joe Faith divides the problem of mind into two general approaches that lead to the "sceptics conclusion" about our ability to know the world. The first via *neuroscience*, illustrated by Descartes argument that if thoughts are generated as neurological events in our head then the knowledge of the contents of our minds will be more reliable than the knowledge of the world, distilled via sense data. The second approach, illustrated via Nietzsche, argues that the validity of the way we judge the reliability of our knowledge changes historically (as discussed in Section 2.2), thereby rendering our current values about our knowledge suspect. This has lead to the break-down in what were previously considered absolute values systems, exemplified by more recent developments in post-modernism, and a general relativism of knowledge systems.

Rorty in *Philosophy and the Mirror of Nature* attempts to counter the view that knowledge is something with objective foundations — that non-material representation or ideas cannot faithfully mirror the external world of nature. For Rorty, the epistemological model pioneered by Descartes and Kant, and the presumption of the ontology of objective truths, is "simply a product of the metaphor of seeing that informs so much of western philosophy" (Jay 1993;

Macey 2000, page 335), knowledge being a matter of conversation and social practice. Hence, truth is seen as a relative construct, dependent on the values of the community from which it is made.

Extreme positions of relativism do not create difficulty in discussions within particular discourses, it is only when disciplines cross boundaries that problems arise, because the relativism breaks down.

3.3 Technology and Interpretations of Realism

Seeing a manufactured image carries the implicit fact that the image was made by someone, and hence the choice of subject, object and framing, even in the most casual of images, is in a sense subjective. This subjectivity is compounded by the act of seeing, which is inherently subjective also. The art critic and cultural historian, John Berger argues that when an image is presented as a work of art, the way people look at it is affected by a series of learnt assumptions, many of which come from classical traditions (or mystifications) promoted by art theory (Berger 1972).

In his essay *Seker Ahmet and the Forest* (Berger 1980, pp. 86-93), Berger recounts his fascination with the painting *Woodcutter in the Forest* by the Turkish artist Seker Ahmet Pasa, who gained an art education in Paris in the nineteenth century before returning to Istanbul where he introduced a “European optic into Turkish art”. Berger argues that the painting is interesting because it shows the artist trying to reconcile two opposed ways of seeing, not trying to simply change his technique (from that of the Turkish pictorial tradition to that of European painting), but to “change his ontology”. This disjuncture is deeper than styles or traditions, and illustrates how the representations of space may be culturally constructed.

Walter Benjamin saw that with the advent of photography, the nature of art was changed (Benjamin & Arendt 1968). Mechanical reproduction changed the way paintings were seen and the realist convictions of painting. Some artists however, realised that the camera had the ability to see beyond that of the eye. Bauhaus artist, Maholy-Nagy wrote:

> In photography we possess an extraordinary instrument for reproduction. But photography is much more than that. Today it is in a fair way

to bringing (optically) something entirely new into the world. (Moholy-Nagy 1967)

Even in photography, the goal of "reproducing the real" was not seen as the pinnacle of this new technology for artists, rather its assets were the expanded understanding and new possibilities the medium potentially made available. One could make a similar argument for image synthesis and computer graphics in general. Indeed, this idea of facilitating an expanded faculty of the imagination was advocated by the philosopher Gaston Blachard in 1958:

> By the swiftness of its actions, the imagination separates us from the past as well as from reality; it faces the future. To the function of reality, wise in experience of the past, as it is defined by the traditional psychology, should be added a function of unreality, which is equally positive... Any weakness in the function of unreality, will hamper the productive psyche. If we cannot imagine, we cannot foresee. (Bachelard 1958)

This "function of unreality" is the crux of the mode of engagement advocated in this thesis. It suggests the distinction made by Aristotle between *historical truth* explaining what has happened, and *poetic truth* explaining the kinds of things that might happen. The use of an interactive computer simulation as a tool for expanding understanding of a process is crucial to both scientific and artistic simulations that operate in this mode of "poetic truth". The physicist Norman Zabusky calls this interaction "computational synergetics" — a process where using computers in "heuristic mode" gives an enhanced understanding of a model, which he claims was previously without precedence (Zabusky 1984).

3.3.1 Postmodern Realism

Contemporary theorists see mimesis take on new definitions because of technology. Baudrillard, for example describes the *hyperreal* as something "more real than real": something fake and artificial that comes to be more definitive of the real than reality itself (such us our understanding of "war" which may come almost exclusively from television). A *simulation* in his terminology is a copy or imitation that substitutes for reality. Television reverses the Platonic relation between mimesis and reality, since the representation precedes the reality and even comes to define it (Baudrillard 1981, 1983; Francblin & Baudrillard 1991).

3.4 Summary: Graphics and Realism

In its quest for photographic verisimilitude, computer graphics uses incomplete assumptions about what it is trying to imitate and relies on implicit cultural interpretations of what constitutes realism in the static and dynamic image. An image synthesised by computer graphics techniques that is indistinguishable from a photograph fulfils the paradox of mimesis: if it is genuinely indistinguishable it is no longer an imitation, but a replication. In general, computer graphics uses a representation of reality that suits both the technical constraints of the medium and cultural assumptions of the classical western tradition of painting and photography. A picture is obviously not a computer, and the world is not symbolic. The world is not like a ray-traced or radiosity image; water is not a collection of infinitely small Newtonian particles; teapots are not really made of parametric, infinitely thin, surfaces. It is unlikely any computer graphics researcher would argue the worldview of the simulation of reality to have a direct and absolute factual correspondence to reality itself (be *isomorphic* in the terminology of simulation).

The goal of this discussion is not to discredit the intellectual effort that goes into the research of computer graphics, with respect to realism. It is to acknowledge that the concept of realism and the mimicry of reality are problematic issues that have a well-explored history in philosophy and the arts. This history is rarely acknowledged in computer graphics, because it works in a domain that serves ideological imperatives that have little need to acknowledge such a history or the problems it raises.

"Realistic" images and simulations are ultimately driven by their usefulness to the domains in which they are employed — engineering and technological sciences, cinematic special effects, interactive games, for example — with emphasis on particular aspects of that realism and the technology that enables it skewed to the particular domain.

Photorealism is an historical movement in art that largely predates its digital counterpart. As an art movement, it is seen to have failed, in that even though the *images* were realistic, visual realism alone is not sufficient for achieving truth in painting. In postmodern philosophies, the concept of absolute truth is problematic, which is one of the reasons Realism's influence has diminished. Photorealistic images were a product of both a technical and conceptual process that could not fulfil the goal that realism had set. In the context of developing an artistic system that is based on techniques from computer graphics, the issue of mimesis takes on a relevance that is more immediate for the artist than the computer graphics researcher.

3.5 Nature

The seemingly innocent word, "Nature", used so often in everyday conversation, belies a raft of rich and complex meanings, "so various and comprehensive in its use as to defy our powers of definition" (Soper 1995).

Since the earliest records of creative expression, artists have sought inspiration from, and reflected upon, nature. Over the previous centuries, the meaning of the term "nature" and the context in which it is used has changed dramatically. Emphasis has shifted from a pure reflection upon form, to a multiplicity of interpretations of natural systems. Key amongst these is the recognition that emergence and process are as much a part of nature as the forms and behaviours they produce.

Reflecting a move away from myth and superstition, our understanding and appreciation of nature has also been shifted in response to scientific theories, notably Darwin's (and Wallace's[29]) theory of evolution by natural selection. More broadly, our theories of nature have come to include the wider processes of the universe as explored by the physical sciences.

However, scientific theories of nature cannot be viewed in isolation. A complex nexus of experience, mediation by media and interpretation of scientific theory, influences the representation of our beliefs about nature. Added to this mix is the increasingly politicised construct of nature, ecology, and culture. Consequently, it is possible to see why artists no longer seek influence from the purity of form alone.

[29] For an engaging discussion regarding the political dynamics of Darwin and Wallace and their theories of evolution and natural selection, see (Quammen 1996).

In the context of this thesis, I am particularly interested in the quality referred to by the poet Manley-Hopkins as *inscape* — the distinctive and essential inner quality of something, especially a natural object or a scene in nature. Inscape cannot be considered through a Kantian framework of inaccessibility, but rather should be linked to the romantic and emotional loss of "traditional" nature.

In this section, I will look at a selection of individual artists (and the art movements they belonged to), as a way of framing the background and heritage of the artistic software system I have developed and described later in this thesis.

3.5.1 Botanical Art

Biological and Botanical art occupies a somewhat special place in the domain of art and science. The European tradition of plant drawing often involved removing specimens from their natural surrounds and drawing them in prepared form, "at times even altering the very substance of the plant" (Blossfeldt & Sachsse 1994). This context is important because it illustrates a tradition of altering and subjectifying aspects of botanical function and structure according to both aesthetic and scientific biases — staging or altering images of reality in order to privilege or detail certain features over others.

Ernst Haeckel's *Art Forms in Nature* is influential in this discussion. Hackel, the first full professor of zoology in Jena, was "renowned for his emphatic advocacy of Darwin's theory of evolution" (Breidbach 1998). Haeckel saw knowledge of nature as *natural aesthetics*, nature having an intangible, Kantian beauty derived from seeing humans and their activities as products of evolution. His argument was a monistic one: human knowledge — subject to and developed within the laws of nature — is in nature itself. Haeckel's perception was determined by a style, seeing nature *à la mode de l'art nouveau* made for a determining approach to his work[30] (Breidbach 1998). Despite his claims of objectivity, Haeckel's "foreign forms filtered through a decorative lens" reveal a perspective in and from his time and culture. He "provided a

[30] This "decorative", ornamental nature of Art Neuveau is seen most clearly in the gregarious ceiling ornament created by Haeckel based on his illustrations of the jellyfish *Toreuma bellagemma*.

picture of nature's entirety, which contemporary physics seemed to loose sight of in its formal operations".

At the end of the nineteenth century, the technology of photography and cinema began to have a major impact on biological and botanical art. Photography's pencil of nature promised a more real representation of botanic forms and an automation of the laborious process of capturing with technical accuracy, the complex detail found in flora's rich structures. As with their drawn predecessors, plant photography often served a dual purpose as images of aesthetic quality and scientific utility.

Karl Blossfeldt's photographs drew inspiration from the pharmaceutical catalogues and classification books of the late Middle Ages and the herbaria of the seventeenth and eighteenth centuries (Adam & Blossfeldt 1999; Blossfeldt & Sachsse 1994). Specimens were placed on a uniform white or grey background, to permit easier comparison, without the distraction of a native milieu. In this case, plants became modernist architectural structures, illustrating nature's feats of engineering (an influential topic of art criticism for the avant-garde artists of the time — 1920s and 30s).

Blossfeldt's photographs typify the use of art within the framework of a semi-scientific context — nature as architect of the elegant and beautiful structure (in a modernist sense). This dialogue persists throughout the twentieth century, evident in key works such as D'Arcy Thompson's *On Growth and Form* (Thompson 1961), Gyorgy Kepes' *Structure in Art & Science* (Kepes 1944), Peter S. Stevens' *Patterns in Nature* (Stevens 1974) and the modern computer graphics analogues lead by Alain Fournier's *The Modeling of Natural Phenomena* (Fournier et al. 1987).

Blossfeldt's philosophy was "part romantic view of nature, part critique of functionalistic design, and part reductive application of Darwinism to both social and aesthetic developments" and his photographs and the styles that were influenced by them all "demonstrate the dubious success of a way of thinking which has constructed a view of nature bent into mechanical shape" (Blossfeldt & Sachsse 1994).

It is also interesting to compare Blossfeldt's photography to that of Imogen Cunningham, another artist well known for her sensuous photographs of plants, who considered the paradox in her pictures; even though objectively rendered, they were produced by "sensation and emotion" (Lorenz & Cunningham 1996). Here manipulation took place to intimate the sensuality and emotive qualities, in contrast to the functionalist emphasis of Blossfeldt's photographs. Likewise in the sensuous plant photographs of Edward Weston, he did not wish to impose his personality upon nature, but to make visible aspects of nature that were "a revelation":

> Through photography I would present the *significance of facts*, so they are transformed from things *seen* into things *known*. Wisdom controlling the means — the camera — makes manifest this knowledge, this revelation, in form communicable to the spectator. (Weston 1973, page 241)

Key amongst these forms and practitioners of botanical and biological art is the concept of nature possessing an inherent beauty that is born out through the emphasis of subjective features. In Blossfeldt's work, particularly, natural structure is shown as a source of architectural beauty — the "bending" of nature into mechanistic form. However, this structure is a static one, frozen by the drawing or photograph in a way that represents a Platonic idealisation of form in a temporal sense. This could be understood as a response to the search for an inscape that melded Darwin's theory of evolution with the technical and industrial economies of the time. Likewise, Cunningham's aesthetic of the hidden and personal space of flora by treating the image as an expressive rather than representational medium, even though the subject matter is "real".

3.5.2 Nature as a Source of General Systems

In the context of the artistic methodology presented in this thesis, there is a shift brought about by the general philosophies of the *system sciences*: systems theory, cybernetics, and artificial life. These fields emphasise the abstracting of process: a shift from *materials* to *mechanisms*. Hence, with the advent of computers and mathematical theories of plant development, there is a shift from formalist architectural beauty alone to a *dynamic* or *algorithmic* beauty (Flake 1998; Prusinkiewicz & Lindenmayer 1990): the re-mediation of *process* via computer simulation.

Nature has served as inspiration for many different artists, but it has also provided, in a dynamic process-based sense, a rich palette for designers.

In the work of Universalist, R. Buckminster Fuller, nature is both a problem to be solved and a source of inspiration in seeking a purity and utility of form (Figure 3-1). Fuller's exploration of mathematical forms in geometry was not unique, but his oeuvre indicates a reflection on form that went beyond the norm for individual disciplines. Fuller's success was in his combination of mathematical form, natural beauty, architectural utility and Promethean thinking (Krausse & Lichtenstein 1999).

This methodology of "systems thinking" involves looking at structures, processes and inter-relationships between them. Educational realisations of this methodology could be found in the Generative systems program at the School of the Art Institute of Chicago or the Conceptual Design Program at San Francisco State University (Wilson, S. 2002, page 336).

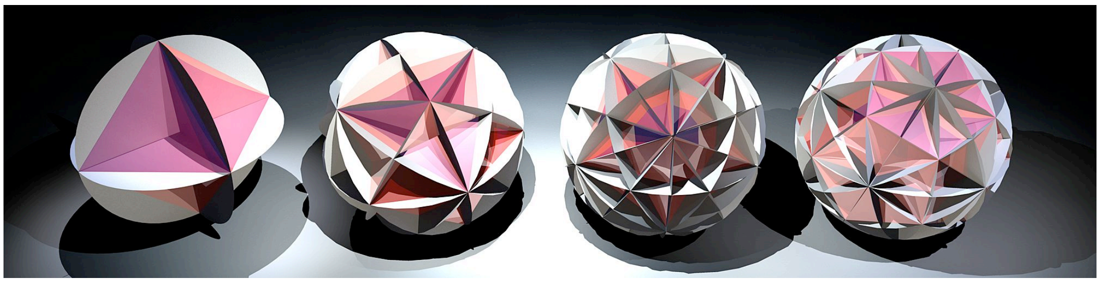


Figure 3-1: *Homage to R. Buckminster Fuller.* Study of geodesics formed by the intersection of great circle planes with the edges of platonic solids. The geometric models were created with the modelling system described later in this thesis.

3.6 Generative Art

This section discusses *generative art* — a specialised category of art that relies on the formal specification of some form or process in order to generate the artistic work. A more formal definition will be given in Section 3.6.2.

3.6.1 Historical Precedents for Generative Art

The term "generative art" refers to a number of distinct, but related artistic practices, so it is often used in many contexts. The earliest adoptions of computer technology for art were necessarily generative, since there was little in the way of pre-configured software, hence paradigms for the simulation of traditional media. Artists had to write their own software in order to generate the results. Undoubtedly too, the fields of systems theory and cybernetics had a major influence on generative art (Bertalanffy 1968; Rosenberg 1983; Wiener 1961), since they looked at systems and information in ways which were novel at the time. Interestingly, these formative years of computing and cybernetics spawned many theories of generative process.

In the 1950s, several researchers attempted to apply the information theory of Shannon and Weaver to aesthetics, believing it could offer a better understanding than psychology. The idea mooted that such a theory of *information aesthetics* could be used for both analysis and synthesis. German philosopher, Max Bense developed a theory of *generative aesthetics*, linking information theory with the generation of aesthetic artefacts and semiotics. This theory was highly influential for a number of artists in the "Stuttgart school", such as Georg Nees[31], Frieder Nake and Manfred Mohr, who were interested in the generative power of algorithms to make art (Nake 1998).

In the late 1960s, Jack Burnham published a series of seminal writings on process-based art. His *Artforum* article, "Systems Esthetics", attempted to define a new role for sculpture — a "paradigm shift" in the terminology of Kuhn — where art could borrow from new discoveries in science and shift from being object-oriented to systems-oriented (Burnham 1968b). These ideas were further articulated in Burnham's book, *Beyond Modern Sculpture*, which predicted movements in generative, robotic, and cyborg art, and related them to historical and contemporary art practices and movements (Burnham 1968a). In a narrative reminiscent of the Constructivist's faith in social change through science and technology, Burnham had faith that the systems esthetic offered a new and superior path for art. This path would follow in the philosophies of science and technology, particularly systems theory and cybernetics.

In a subsequent issue of *Artforum*, critic Terry Fenton responded to this new romance (or "muddle" as it was called) between art and science. Fenton was critical of movements that tried to use science as a methodology, or

[31] According to Frieder Nake, Nees wrote one of the first Ph.D. theses on generative computer art in 1969, its title (in German) translates to "Generative Computer Graphics". "In a very trivial sense, what was meant was nothing but the application of algorithms to produce drawing[s] as aesthetic objects" (Nake 2002).

worse, a philosophy for art, particularly those of Burnham and Gyorgy Kepes, arguing that art should not become "a handmaiden of science as technology is a handmaiden of science" (Fenton 1969). In the years that followed, the systems esthetic did not become the dominant paradigm for art.

Writing some twenty years following *Beyond Modern Sculpture*, Burnham acknowledged that the social utopias for art drawn from science had not arrived. He wrote that "most computer artists became profoundly disillusioned with the creative potential of tools" and that "the use of computers in the arts has yet to produce anything approaching an entirely new aesthetic experience" (Burnham 1986).

Looking back, almost twenty years on again from Burnham's concession about the failure of art and technology, a new appraisal is possible.[32] While the effect of science and technology on the "art world" may be oscillating, and in some sense peripheral, the cumulative effect of the "articism" of technology that began in university labs and artists studios in the 50s and 60s has had a profound effect on mainstream technological culture. Hence, much of the innovation today is not achieved within the precious bubble of fine art, but by those who work in the industries of popular culture — computer graphics, film, music videos, games, robotics, and the Internet. This innovation may be problematic from a critical perspective, but suggests a cultural mutation of the ideals (although not necessarily the ideas) and programs established by Bense, Moholy-Nagy, Burnham, Kepes and others.

Burnham's vision for systems theory and cybernetics in the art of the 1960s, sees a curious echo in the artificial life art and generative art of the twenty-first century. It therefore serves as an important historical legacy for many artists working in this area today. However, in the spirit of Fenton's original criticism, it is legitimate to question if these forms of art have indeed become the "handmaidens of technology", serving the technological interests of modern industrial and information cultures. Perhaps the truth lies closer to a collaborative or synergetic relationship with the rationalism of the engineer with the techno-romanticism of the new media artist (Coyne 1999), in a *push-me-pull-you* entanglement mediated by the commerce and glamour of technological and creative industries.

[32] For alternate views on the legacy of Jack Burnham in new media art see (Penny 1999; Whitelaw 1998).

3.6.2 Definition of Generative Art

A contemporary definition of generative art involves the use of biological metaphors (Dorin & McCormack 2001). The terms *genotype* and *phenotype* are used to represent the distinct aspects of this process. Essentially the authoring process is directed towards the creation of the genotype. The genotype is a formal specification of process, generally unambiguous. When this process description is enacted, it generates the phenotype, which is effectively the experience of the artwork. Figure 3-2 illustrates these key elements. An important factor in this generative process is that through the processes specified in the genotype, the phenotype "unfolds in the world". In informational terms, this means that the volume of information generated in the phenotype is significantly greater than the genotype itself (often referred to as *database amplification* (Smith 1984)). It is through the application of a generative process that this amplification occurs. In some situations, the genotype will include specification of generative processes that may act on the genotype itself — leading to new outcomes in the phenotype. Physical and/or computational interaction may also be possible between elements.

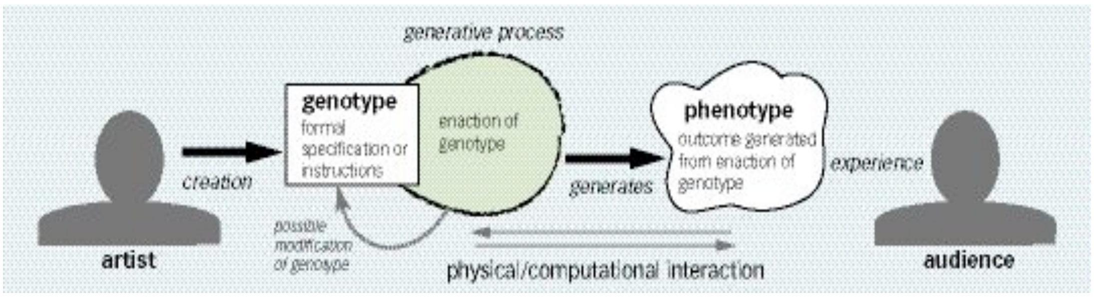
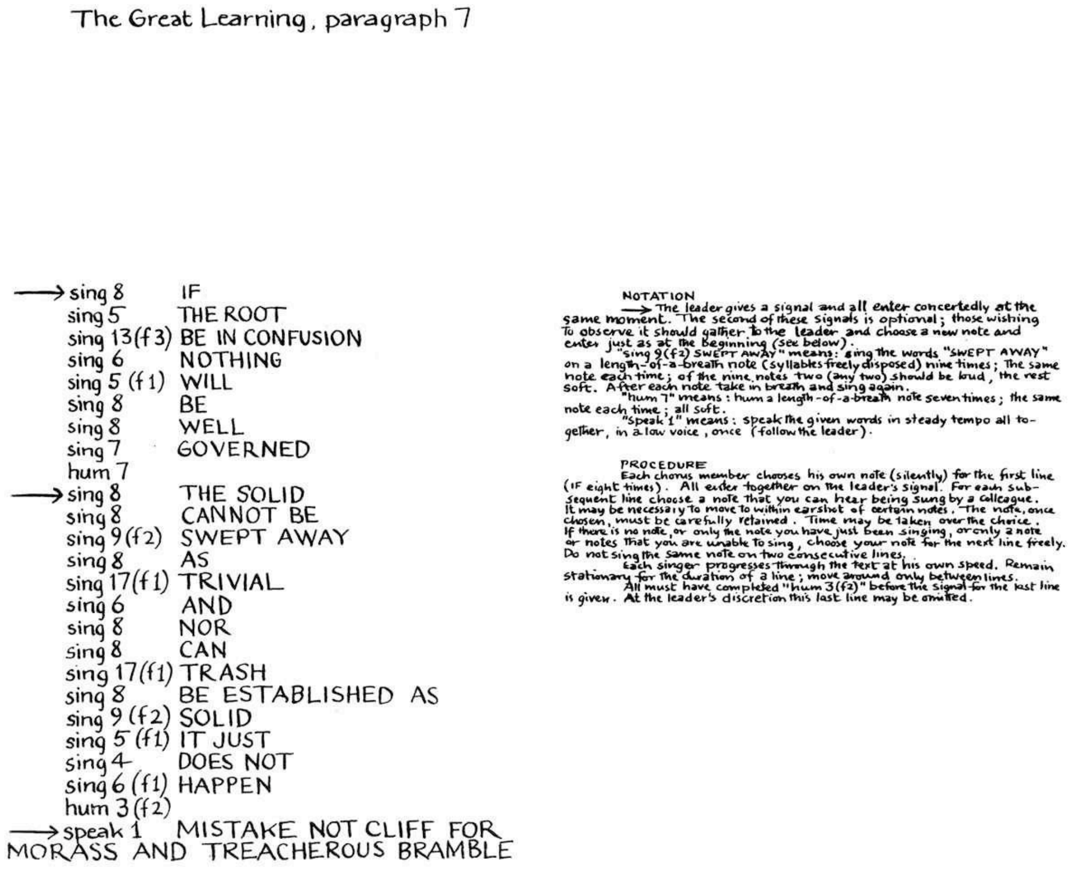

generative process
genotype
formal specification or instructions
enaction of genotype
phenotype
outcome generated from enaction of genotype
creation
generates
experience
artist
possible modification of genotype
physical/computational interaction
audience

Figure 3-2: Overview of the generative process.

The spatial layout of this diagram should not be interpreted literally (as in: "the artist is separated from the audience by some formal mechanisms", for example). Layers of interaction may occur between all the elements of this diagram, possibly blurring the distinction between artist and audience. People experiencing the work may be part of the generative process, which may include chance or physical events that occur as the phenotype is generated. Hence, these terms are generalisations and should not be considered limiting.

3.6.3 Non-Computer Based Generative Art

Typically, we might think of the genotype (generative process specification) as being a set of unambiguous, mechanical instructions, such as those that could be executed by a Turing machine or digital computer. However, this need not be the case. Consider the *Happening* events of artist Allan Kaprow, such as *Fluids* (1967), where instructions were issued for the building of a series of large rectangular "houses", built from blocks of ice and later left to melt.



The Great Learning, paragraph 7

→ sing 8	IF
sing 5	THE ROOT
sing 13(f3)	BE IN CONFUSION
sing 6	NOTHING
sing 5 (f1)	WILL
sing 8	BE
sing 8	WELL
sing 7	GOVERNED
hum 7	
→ sing 8	THE SOLID
sing 8	CANNOT BE
sing 9(f2)	SWEPT AWAY
sing 8	AS
sing 17(f1)	TRIVIAL
sing 6	AND
sing 8	NOR
sing 8	CAN
sing 17(f1)	TRASH
sing 8	BE ESTABLISHED AS
sing 9 (f2)	SOLID
sing 5 (f1)	IT JUST
sing 4	DOES NOT
sing 6 (f1)	HAPPEN
hum 3 (f2)	
→ speak 1	MISTAKE NOT CLIFF FOR MORASS AND TREACHEROUS BRAMBLE

NOTATION

→ The leader gives a signal and all enter concertedly at the same moment. The second of these signals is optional; those wishing to observe it should gather to the leader and choose a new note and enter just as at the beginning (see below).

"sing 9(f2) SWEPT AWAY" means: sing the words "SWEPT AWAY" on a length-of-a-breath note (syllables freely disposed) nine times; the same note each time; of the nine notes two (any two) should be loud, the rest soft. After each note take in breath and sing again.

"hum 7" means: hum a length-of-a-breath note seven times; the same note each time; all soft.

"speak 1" means: speak the given words in steady tempo all together, in a low voice, once (follow the leader).

PROCEDURE

Each chorus member chooses his own note (silently) for the first line (IF eight times). All enter together on the leader's signal. For each subsequent line choose a note that you can hear being sung by a colleague. It may be necessary to move to within earshot of certain notes. The note, once chosen, must be carefully retained. Time may be taken over the choice. If there is no note, or only the note you have just been singing, or only a note or notes that you are unable to sing, choose your note for the next line freely. Do not sing the same note on two consecutive lines.

Each singer progresses through the text at his own speed. Remain stationary for the duration of a line; move around only between lines.

All must have completed "hum 3(f2)" before the signal for the last line is given. At the leader's discretion this last line may be omitted.

Figure 3-3: Cornelius Cardew: Score for Paragraph 7 of 'The Great Learning' (1967).

British composer Cornelius Cardew's *Paragraph 7* of *The Great Learning* is a generative musical work for a chorus of singers (Cardew 1971). Each performer is provided with the same set of instructions (the genotype, shown in Figure 3-3). The instructions consist of a series of words and phrases to be sung on the "length-of-a-breath" note. The performers begin by selecting a note to sing at random, but are then given instructions such as "choose a note that you can hear being sung by a colleague" and "do not sing the same note on two consecutive lines". *Paragraph 7* also highlights another common feature of generative systems: the emergence of new properties that result from local interactions between individual components. These new properties are not specified in the genotype — they *emerge* via the generative process.

A more recent example of generative art that is not computer based can be found in the work of the French artist, Hubert Duprat. In a series of sculptural works, Duprat works "in collaboration" with caddis fly larvae, replacing their natural environment with a series of artificial environments of gold and precious stones (Duprat & Besson 1998). The caddis fly larvae construct the sculptures from the materials of these environments (the casing made by the animal required during metamorphosis), resulting in tiny cases of gold, opals, sapphires, etc. In this case, the artist intervenes on an existing natural process to generate the artwork. Duprat's work draws attention to the relationship between organism and environment, a relationship unexplored in many computer-based works.

3.6.4 Computer Based Generative Art

Naturally, the computer seems the ideal vehicle for making generative art because of its ability to enact formal processes automatically. Computers were not designed as "art-machines", but they have been co-opted to the task particularly well in the case of generative art. The idea of using machines or automata to make art is not new, and dates back at least to the ancient Greeks.

When considering examples, it is important to make a distinction between machines that are generative by the definition in Section 3.6.2, and those simply used as "playback" devices. Machine processes are considered generative in the sense that they are designed to create new works from the genotype, in contrast to the many other mechanical systems that have been made for the purposes of recording and replaying performances — where the output is largely predictable and duplicates a performance made elsewhere or by other means.

3.6.4.1 A-Life Art and Generative Art

The term "A-life Art" or "Artificial Life Art" refers to artworks created based on the philosophies and research from Artificial Life (Adami 1998; Boden

1996; Langton 1989), using systems such as cellular automata, L-systems, neural networks, self-organising systems and evolutionary algorithms to create generative artworks, often with the goal of simulating life-like behaviour (also a goal in A-life research).

In relation to generative art, A-life art can be considered a “specialised” subset of generative art as a whole, in that generative processes are used in order to portray some “life-like” behaviour in the phenotype. Generative art, explicitly or implicitly, references natural phenomena (such as emergence) or natural processes (such as evolution), as does artificial life.

3.6.5 The Computer in Visual and Screen-Based Art

As discussed in Section 3.6.1, early computer art was necessarily generative. John Whitney Sr. was one of the first artists to see the potential of the computer for the creation of screen-based visual art. His first computer work, made in the 1950s using an M-5 anti-aircraft gun director was converted into an analogue computer and used to make films of dynamic, organic patterns based on mathematical equations (Jankel & Morton 1984). Together with his brother James, Whitney produced a series of remarkable animations, beginning first with “painting-on-film” techniques that served as the aesthetic basis for later computer generated works (such as *Lapis* (1963-66)). Subsequent films included *Permutations* (1967), made while an artist at IBM, and what he considered his masterwork, *Arabesque* (1975) were both created on digital computers. Whitney used the metaphor of “writing on water” to succinctly describe his process of dynamic visualisation. Throughout his career was interested in developing new “languages” that related algorithmic image and sound. His statement, “Above all, I want to demonstrate that electronic music and electronic color-in-action combine to make an inseparable whole that is much greater than its parts,” underlies his integrated approach to the emergence of “visual music” in his art.

Also in the 1950s, American artist Ben Laponsky used analogue computers to create artworks realised in a variety of media, including photographs, films, light-boxes and kinetic oscilloscope displays. Laponsky used methods based on his experience with mathematically defined geometric systems, such as harmonograph machine tracings, to program a from of oscillography (Laposky 1975).

Charles Csuri, another computer art pioneer, developed a number of generative techniques beginning in the late 1960s, creating animations and still images using those techniques.

The German artist Manfred Mohr began as a painter, being interested in formal Constructivist art, particularly the influence of Jazz music expressed in a visual domain. He later turned to fractured cubes and then n-dimensional cubes, describing the structure of the cube as a "system" or "alphabet". His cube-based works were generative, in that Mohr specified the algorithms that generated new structures and relationships in their output. He describes the results as being "unique to the computer".

3.6.6 Related Artificial Life Art Systems

Yoichiro Kawaguchi was a pioneering artist in the development of computer graphics software to facilitate his art (Kawaguchi 1985). His "GROWTH" (Growth Rationale Object Work Theorem) system was inspired by the work of D'Arcy Thompson, and permitted the morphogenic modelling of a variety of natural shapes and forms, such as shells, horns and plants. He used an iterative, rule-based algorithm to generate dynamic growth in a range of branching structures (Kawaguchi 1982). Following this, he refined his branching structures to include structures formed using density distribution ("Metaball") techniques run on the LINKS-1 system developed by Nishimura et. al. (Nishimura et al. 1983; Omura, Kawaguchi & Noji 1985).

American artist and researcher Karl Sims applied evolutionary techniques to the generation of plant-like structures for his film *Panspermia* (Sims 1990a). His techniques for interactive evolution (Sims 1991b, 1991a, 1993), based on the original ideas from Richard Dawkins' *Biomorph* software, described in (Dawkins 1986), were strongly influential for both artists and researchers. They are discussed further in Chapter 10.

Jules Bloomenthal developed modelling techniques to model branching structures using generalised cylinders defined over branching topologies. A free-form surface model was used to connect the branching limbs and "blobby" techniques for the tree trunk. He used these techniques to create a convincing visual model of a maple tree (Bloomenthal 1985). Peter Oppenheimer used recursive techniques to develop "fractal" plant and tree models for creative purposes (Oppenheimer 1986, 1988). His models produced strange and complex branching models, which were used for both still images and animation.

William Reeves and Ricki Blau developed stochastic shading and rendering algorithms, based around particle systems, to create "impressionistic" landscapes of grass and trees. Unlike the models of Bloomenthal and Smith (Smith 1984), which focused on the detailed structure of individual plants, this model took a "global view of the forest environment" (Reeves & Blau 1985) and produced softer impressionistic images of forests and vegetation, used in a number of early Pixar (then Lucasfilm) films.

William Latham and Steven Todd developed a system to evolve three-dimensional forms based on CSG techniques (Todd, S. & Latham 1991). In addition, Latham also attempted to place historically his artistic practice in terms of a distinctive artistic style, which he called *evolutionism* (Todd, S. & Latham 1992). Latham's main emphasis was on the synthesis of form, the focus on process implicit in the techniques used to produce his art.

Midori Kitagawa De Leon developed a system, known as "BOGAS" to model branching structures with C^1 continuity using cubic Hermite interpolation (De Leon 1990a, 1991). She produced several beautiful and engaging animations and still images using the techniques she developed (De Leon 1990b, 2002).

The early work of Christa Sommerer and Laurent Mignonneau developed systems that incorporated developmental models. In their work, *Interactive Plant Growing* the physical interaction with real plants causes synthesised plant-like structures to grow on a large video projection screen. This interactive response was further developed in the work *Transplant*, where user's movement was transformed into growth of plant-like objects projected around the viewer (Sommerer & Mignonneau 1998).

Sommerer and Mignonneau's evolutionary work *AVolve* utilised the metaphor of a virtual fish tank. This work defined two distinct processes for users. The first used a "conventional" interactive design metaphor, where via a touch screen, profiles and characteristics of the creature's appearance and behaviour were specified. Following this stage the interaction switches to a more poetic context, where users are asked to interact with the virtual creatures by moving their hands in a pool into which images of the creatures are projected. This work illustrates an interesting duality whereby the mode of interaction signifies the implicit assumptions of the artists — in the design mode the interface is a technical one, similar to standard CAD tools. At this stage, the user is not required to accept the creatures as "living", rather machines designed by CAD–like techniques. Later however, when the creatures are "released" into the real pool, the user is implicitly asked to accept that these are living creatures and the mode of interaction changes from a technical and functional mode to a more playful anthropomorphic one.

3.7 Summary

There are many examples of generative art and the works discussed above have been necessarily selective. Today, a number of conferences are devoted to both critical, aesthetic and technical aspects of generative art, including the *Iteration* series of conferences, organised by myself and Alan Dorin in 1999 and 2001 (Dorin 2001b; Dorin & McCormack 1999) and the Generative Art conferences held annually in Milan, Italy.

A brief history of generative art has been presented as a basis for understanding contemporary approaches that make use of computational processes. In the context of the systems presented later in this thesis, I have also introduced some key issues in relation to botanical and systems art, because in many ways these art forms are trying to "copy nature" using photography and associated technologies. I have also discussed the difficulty in using photorealistic computer graphics as the basis of an artistic process or representation scheme, particularly if one wishes to seek a "poetic truth" in any representation.

4 Emergence[33]

Every doctrine of aesthetics, when put into practice, demands a particular mode of expression — in fact, a technique of its own.

—Igor Stravinsky, 1936

In the previous chapter, the area of generative art was introduced as an appropriate artistic paradigm under which to classify the artwork developed using the software systems described later in this thesis. This chapter provides a survey of critical and technical issues of relevance to generative art. In particular, it examines the concept of *emergence*, looking at its historical origins, interpretation in different disciplines, and salient issues surrounding its classification and meaning for developing generative art. These issues include the hierarchy of levels associated with emergence, recognition and ontology of patterns, prediction, and determinism. Each of these topics are then related to attempts to create with computers emergent phenomena for artistic purposes. Several methodologies for developing emergent generative art are discussed including what is termed in this chapter "the computational sublime". This definition is considered in relation to historical and contemporary definitions of the sublime and is posited as a methodology for thinking about the process of creating an artwork that is "more than the sum of its parts".

4.1 Introduction

As with a number of art movements, generative art draws from selected elements of science and philosophy as part of its basis, and as a primary influ-

[33] Sections of this chapter were first published in (McCormack & Dorin 2001). The material that appears in this chapter is exclusively the original work of the first author.

ence on its motivation. Naturally, these influences are well known and widely discussed in scientific literature and from scientific perspectives. However, little attention has been paid to these influences from within generative art beyond the fact that they are seen to be part of science's way of describing the world. If such influences are important to the art form, they need to be addressed from the perspective of generative art itself.

In particular, I will examine the concept of *emergence*, with a view to using it as a basis for developing strategies in generative art. The views presented here are influenced primarily by investigations in evolutionary biology, philosophy of science, cybernetics, systems theory, and artificial life. These frameworks are not the usual basis for forming a discussion about art, but for generative art, they hold special significance, often being the major foundations for developing artworks and forming ideas. This is partly due to the nebulous influence of concepts such as emergence, novelty, chaos theory, determinism, complexity, self-organization, and "natural" selection on generative works. If these are going to form part of the foundation of a practice, it is surely wise to ensure they are well constructed — lest whatever is constructed on them may collapse.

No doubt, there are other important criteria for thinking about generative art. The purpose here is to examine the concept of emergence and propose possible strategies that the generative artist can use to exploit it. At the very least, this may provide a way of creating and, in parallel, critically evaluating generative art.

How might we think about generative processes in relation to an artistic practice? Oddly enough, this question has been asked many times before in relation to cybernetics and artificial life. Often in these disciplines, less importance is attached to distinctions between art and science than is attached to the philosophy used to approach the endeavour (Risan 1997).

4.2 Artificial Life

The idea of giving life to inanimate objects is a consistent fascination for humans. Domains of enquiry such as artificial life (AL) suggest by their very name a kind of "Frankensteinian" fascination with mortality, a perceived reversal of entropy, and the super-human ability to breathe real life into inanimate objects. It appears that "giving life" to inanimate objects interests us, because "life" interests us. Such modalities may be either conscious or unconscious forces in the AL artist's creative impetus.

Different interest groups have different interpretations and visions for Artificial Life (*Artificial Lives* as Bonabeau and Theraulaz call them (Bonabeau & Theraulaz 1994)). The perspective presented in this chapter is taken from that of the artist wishing to engage critically with AL. Therefore, important and cognisant ideas that compel the AL artist may centre, for example, on the concepts of control, inscape, the sublime, novelty, aesthetics, phenomenology, determinism, causation, and ecology.

When we express our relationship to "the natural" through poesis[34], explicitly or implicitly we express our concern about *control*. Nature is seen as a force that must be controlled, harnessed and tamed. This belief is reflected in popular notions of nature as "the chaos", the uncontrollable force, and is exemplified by its effects and their consequences (death). For example, the act of gardening is often quoted as a metaphor to describe aesthetic evolution.[35] In some sense, gardening is about mastering the uncontrollable — harnessing nature and manipulating it for aesthetic purposes (from the perspective of the gardener). The issue of control translates from the biological garden to the digital garden; in the case of aesthetic evolution, it becomes even more acute — the digital gardener selects what will "live" and what will "die".

Important also, is AL's original claim of broadening the definition of life and offering new or novel forms of life not currently observed in terrestrial biology — *life-as-it-could-be* (Langton 1989). Much of the life-as-it-could-be mode of investigation has been dismissed by scientists because it is ill defined (Bonabeau & Theraulaz 1994) — if we were to create life-as-it-could-be that was significantly different from *life-as-we-know-it*, how would we recognise it as life? It is also easy to misinterpret life-as-it-could-be as simply a search for the novel or bizarre — how far can the phenomenological experience of life

[34] Poeisis is the process of bringing-forth via human hands, of revealing the world in a way that could not have occurred by natural processes (which are the processes of *physis*).

[35] Evolutionary artist, William Latham says "The artistic process takes place in two stages: creation and gardening. The artist first creates the systems of the virtual world…the artist then becomes a gardener within this world he has created;" see (Todd, S. & Latham 1992).

(particularly in an art context) be (un)reasonably extended in a postmodern view of the world?

Indeed, AL techniques form part of the broader category of generative art — art that uses some form of generative process in its realisation (introduced in the previous chapter).

4.3 Generative Art

A definition of generative art was given in Section 3.6. Generative art practice focuses on the production and composition of the genotype and the media in which it produces the phenotype. When run, interpreted, or performed, the genotype produces the phenotype, the work to be experienced and the realisation of the process encoded by the genotype.

Generative art usually involves poeisis, which suggests that it should reveal the world in ways that nature can't, hence technology seems a possible (though as we have seen, not necessarily unique) vehicle to achieve this aim. Implicit too, is the act of creation, but it is poignant to ask in a critical context what is being created, what is being revealed, and what is the difference between the two.

The role of the artist in developing a generative artwork often involves creation and manipulation of the genotype and the developmental and (pseudo) physical process systems that "unfold" it into the phenotype, "revealing" it in the world. Creation needs to involve some form of novelty, addressing the difference between creating and revealing in this context.

4.4 Emergence

In this section, one of the central concepts for developing and understanding generative art is examined: *emergence*. Emergence is an all-encompassing term, with a nexus of barely related meanings in different domains, making it a difficult term to clearly define, let alone understand. Since the term's early use, almost every author has provided his or her own sub-categorisation for different types of emergence.[36] There is little consensus between individual

[36] One recent survey distinguishes no less than 27 different classifications (McDonough 2002).

authors, much less between disciplines. Debate continues as to the merits of the concept in a number of areas, primarily trying to decide if emergence is a linguistic, epistemic or ontological construct. Comprehensive overviews and historical reviews can be found in (Beckermann, Flohr & Kim 1992; Blitz 1992).

The common non-specialist interpretation of the term *emergence* refers to revealing, appearing, or "making visible" an event, object, or the outcome of a process. In a creative context, emergence also encompasses novelty, surprise, spontaneity, agency, even creativity itself — aspects of emergence I will examine more formally in this section.

4.4.1 History and Overview of Emergence

Emergence has its origins in the nineteenth century studies of physical, chemical and biological systems. John Stuart Mill drew a distinction between "two modes of the conjoint action of causes, the mechanical and the chemical," (Mill 1872). Influenced by Mill, George Henry Lewes recognised Mill's fundamental differences between heteropathic and homopathic effects, calling them *emergents* and *resultants* respectively (Lewes 1879). As described by C. Lloyd Morgan (Morgan 1923), *emergent evolution* (*emergentism*) describes the "incoming of the new", that is, emergence is defined as the *creation of new properties*. Emergentism was a philosophy about the nature of the universe and the way material elements combine to make structures of increasing complexity. When the complexity of material configurations reaches a certain level, genuinely novel properties emerge that have not been instantiated before and could not have been predicted (Beckermann 1992).

The oft-quoted example of Mill's (from Morgan) relates to molecular chemistry: carbon has certain properties, sulphur has certain properties; when the two are combined the result is not an additive mixture of the two but a new compound (carbon disulphide), some of the properties of which are quite different than those of either component, hence the interpretation that *the whole is more than the sum of its parts*. Morgan, referencing the work of Mill, Lewes and psychologist Wundt's "principle of creative synthesis", saw emergence as a phenomenon that occurs in many different systems or *hierarchical levels* including molecular interactions, life, mind, and self-consciousness. This concept of emergence has been described as "that reasonable aspect of vitalism which is worth to maintain" (Emmeche, Køppe & Stjernfelt 1997), that is, it removes vitalism's non-materialist suppositions.

Emergence and emergentism have continued to rise and fall in popularity throughout the twentieth century. Important criticisms in Nagel's *Structure of Science* (Nagel 1961) and Hempel and Oppenheim's *Studies in the logic of Explanation* (Hempel & Oppenheim 1948) saw support for emergence as a strong philosophical concept wane for many years. A key criticism of emergence as a phenomenon is that its usefulness as a classification method is limited because it tends to ignore the specific physicality of the individual systems — how similar is the emergence of new properties of carbon disulphide from its atomic components to the emergence of consciousness from cells? How does giving a property the attribute of *emergent* aid in our understanding of that phenomena?

The idea, which is nascent in emergentism, of separating phenomena and processes from substance became formalised in the systems-theoretic approach of the 1940s. Systems theory manifested itself in a number of areas, such as the philosophy of Bunge, Bahm, and Laszlo; information theory of Klir; biology of von Bertalanffy, Thom and Waddington, cybernetics of Weiner, Ashby and Rosenberg (Audi 1999, page 898) and today in artificial life.

4.4.2 Levels and Patterns

Emmeche, Køppe and Stjernfelt (Emmeche, Køppe & Stjernfelt 1997), give a detailed epistemic analysis of Morgan's "creation of new properties". They differentiate between three different uses of the word "properties" — referring to *primary levels* (similar to Morgan, the borders between the major sciences), *secondary levels* (sub-fields within the major sciences), and aspects of *single entities*. These classifications suggest different types of emergence. In relation to the creation of levels, their *gestalt view* holds that the higher level manifests itself as a *pattern* or as a special arrangement of entities at the lower level.

The emergence of patterns is of concern to a number of authors. In the field of *emergent computing*, for example, Forrest writes:

> In these systems interesting global behavior *emerges* from many local interactions. When the emergent behavior is also a computation, we refer to the system as an *emergent computation*. . . . Three important and overlapping themes that exhibit emergent computation are self-organization, collective phenomena, and cooperative behavior (absence of any centralised control). (Forrest 1990)

What Forrest terms “interesting global behaviour” suggests two important issues in understanding emergence: (1) that of the perceptible, and (2) that of the influence of the observer. As Forrest admits:

> The emergent phenomena of interest are often understood implicitly rather than explicitly. Currently, many emergent computations are interpreted by the perceptual system of the person running the experiment.

How do we implicitly recognise “interesting” patterns? What is the difference between emergent and non-emergent patterns? Philosopher Daniel Dennett discusses the “reality of patterns” in (Dennett 1991). Looking at how agents distinguish a particular pattern from noise, he makes important observations about the information-theoretic content of a pattern in relation to its ontological status. Visual systems evolved to distinguish pattern from noise, but the “level” of pattern recognition has evolutionary constraints: a balance between the fidelity of pattern perception, its costs and payoffs. If we rely on recognition of patterns to justify emergence in systems, there could be patterns that we as observers cannot recognise[37], yet may still be “interesting” when studying emergent phenomena.

Information-theoretic approaches to understanding patterns in emergent systems have been studied (for example) by Crutchfield (Crutchfield, J. P. 1994). He recognises that “patterns are guessed rather than verified” and so seeks information-theoretic measures to obtain a more objective analysis of pattern formation. Crutchfield defines *intrinsic emergence*, where the system itself capitalises on patterns that appear (i.e. the patterns exploit their own dynamics).

Our natural perception[38] defines the concept of everyday things and objects (animals, plants, etc.) that we have evolved to perceive at a particular level, in order to function in the world. Science has added to this a means for us to

[37] Dennett often uses Wimsatt’s example that an ant-eater sums a collection of ants to their totality, hence “sees” them a whole rather than as a collection of individual ants (Wimsatt 1980).

[38] Meaning unaided by technology.

"see" at other levels. Following on from Dennett, taking perception to its ultimate end, if we could "perceive" the universe in a kind of gods-eye view or "Laplacian inversion", with the recognition of every single sub-atomic particle over its spatio-temporal configuration, would we have any need for patterns or levels?[39] These concepts (patterns and levels) are necessary conveniences, developed as the result of pragmatic evolutionary pressures, to assist our survival within the limitations of being perceptual entities that are part of the world.

Compare to this, the view that it is impossible to interpret a lower-level explanation without using some higher-level concepts to identify what is going on. That is, higher-level phenomena need to be recognised as a basis to identify what must be explained at the lower level (Kincaid 1988).

4.4.3 Prediction, Explanation and Determinism

Life can only be understood backwards; but it must be lived forwards.

—*Kierkegaard, quoted in (Dennett 1984)*

Since the coining of the term *emergence* by Lewes (McLaughlin 2001), a distinction is made between those emergent properties that are explainable as products of lower level interactions, and those that are not. The meaning of "explainable" is the crux of the issue and the basis of the reductionist/emergentism debate. Emmeche et. al. refer to two kinds of processes: those that cannot yet be explained but are not, in principle unexplainable, and processes that are in some sense of the word, unexplainable (Emmeche, Køppe & Stjernfelt 1997). It is this second sense that usually provokes the reductionists into retaliation, because this implies that such processes are ontologically irreducible. Further, how are we to know what *will* be explainable in the future? In this case, it becomes impossible to distinguish between the epistemological and ontological senses of emergence.

Consequently, the idea that "the whole is *more* than the sum of its parts" may be expanded to give *form* its own ontological status[40] — the term "more"

[39] Such a fanciful proposition raises numerous difficulties, particularly given the fundamental uncertainty in measurement of sub-atomic particles. However, the proposition can be more appropriately applied to computer simulations.

[40] That is, non-subjectivist features.

defining a “specific series of spatial and morphological relationships between the parts”. Matter and form, “opposing but not contradictory points of view of the same reality” (Emmeche, Køppe & Stjernfelt 1997).

Central to a modern concept of emergence are the relevance of *determinism* and boundary conditions. Von Neumann pointed out that physical laws are reversible in time, but that measurement is intrinsically irreversible (Pattee 1988) — (Prigogine & Stengers 1984) offer a contradictory view. Modern physics has shown that even those systems that can be described deterministically are subject to a critical dependence on initial conditions (for example, the *three body problem*). Polanyi recognised that while physics may be able to describe what is going on at a micro level, the macro emergent properties cannot be predicted from the micro level physics, because they are computationally irreducible — determined by boundary conditions at the macro level (Polanyi 1968). That is, the lower level laws are *unspecific* (Bonabeau & Theraulaz 1994) with respect to the higher-level phenomena they may produce. Emmeche et. al. use the example of the cell to illustrate this idea: “if you list all known chemical regularities and laws, it would be impossible for you, on the basis of this list and without any knowledge of the biological cell, to select those entities, regularities and types of behaviour which are specific to the biological cell” (Emmeche 1994).

Thus, according to these views, Physics presents an immense phase space of possibilities, in which it is impossible to determine exactly *what* will emerge at higher levels. Emergence can only be recognised *after* it has occurred, since it cannot be predicted in principle.

4.5 Emergence for Generative Art

> Most electronic artists are looking for an out-of-control quality that will result in their work actually having outcomes that they did not anticipate. If the piece does not surprise the author in some way then it is not truly successful in my opinion.
>
> —*Rafael Lozano-Hemmer quoted in an interview with Heimo Ranzenbacher. (Stocker & Schöpf 2001)*

The richness of emergence in the physical world serves as a great source of inspiration to generative artists.[41] Artists are often looking for surprise, novelty, agency and that "out-of-control" feeling in their work, what Ashby describes as *Descarte's Dictum:* how can a designer build a device which outperforms the designer's specifications? (Ashby 1956)

Artists can get away with much more than scientists can where emergence is concerned, since art is not bound by the same obligations as science. But in gaining such freedom, the artist also acquires new problems because, in general, the search space lacks *reference points*[42] and becomes potentially vast. Given that true emergence defies prediction, how can one begin to *design* works that *do* have outcomes that were not anticipated?

Langton's *life-as-it-could-be* seems like an excellent starting point for developing the concept of *art-as-it-could-be* — emergent creative behaviour in artificial systems. However, given the problems discussed in previous sections, achieving life-as-it-could-be seems difficult, life-as-it-could-be creating art-as-it-could-be even more so. Moreover, if we are going to find it tough to recognise life-as-it-could-be, surely art-as-it-could-be will be unrecognisable, incomprehensible, or just plain uninteresting.

The idea of autonomous systems making art might seem appealing, but how does it relate to our understanding of human art? Bowerbirds might be considered autonomous systems that make "art", but such activities remain principally of interest to biologists, not art critics. The creation of evolving agents that develop their own artistic practices should not be confused with the goal of widening the scope of art for human appreciation.

4.5.1 Creating Emergent Art with Machines

"But you know, all pictures painted inside, in the studio, will never be as good as the things done outside."

—Paul Cézanne in a letter to Emile Zola, 19 October 1866

[41] From here on, when we refer to "art" and "artists" I am primarily referring to generative art and those who make it.

[42] By reference points, I mean events, relations and epistemologies that form the basis for developing a particular work. Science has the physical world and the scientific method as possible reference points. Normally, conventional artists will have their own personal reference points and those from art theory, but with works that attempt genuine emergent properties, these may not be appropriate, particularly if the artist does not wish to simply mimic the existing reference points of art or science.

Today, generative art is often implemented on a computer.[43] Thus it is poignant to consider the limitations and possibilities of computation for creativity, and of the computer as an "art machine" if we want to create emergent works. As discussed in Section 4.2, some of AL's goals (e.g. emergent behaviour) are not dissimilar and there is much in biological and AL literature discussing the limitations of using symbol processing machines to make strong AL, for example. This could be a suitable starting point for examining similar issues in generative art.

Pattee distinguishes between *simulation* — as a metaphorical representation of a specific structure or behaviour that we recognise as "standing for" but not realizing the system being simulated (weak AL) and *realisation* — a literal, substantial replacement of that system (strong AL) (Pattee 1988). Computers are symbol-processing machines and while they are capable of simulating physical systems and phenomena, a symbol processing simulation is not a priori grounds for a theory of what is being simulated.

This raises an important question in relation to computer-based generative art works. If they are simulations, their basis must come from simulating something known. Most current generative art works are developed in this way (they *tweak* the *conceptual space* in the terminology of Boden (Boden 1994)). Since new emergent phenomena cannot be predicted, we must depend on the simulation of known emergent systems, or on intuition and heuristics based on existing systems in order to guess which particular configurations of micro properties will result in emergent macro-phenomena. Computation has a smaller phase space than that of physics, that which is practically computable, even smaller. Increasing computing resources has two simultaneous implications for this practical space: it enlarges its size, giving more potential configurations, and yet it increases the speed at which this space may be searched. This recent advance in search speed often makes searches that would be impractical to explore in physical systems possible in simulation.

This begs the question: is generative art made on computers just tweaking the simulation of existing systems, or is it exploring the phase space of com-

[43] "often" does not necessarily imply the best. Some generative artworks have used other physical entities to set up process-based works with considerable longevity and critical success (see (Dorin & McCormack 2001) for examples).

putation in a genuinely novel way? Should our starting point in developing generative art be based in the simulation of reality, particularly given the discussion regarding the problems of representing the real in art? Alternatively, do we begin with the more constrained possibility of exploring the symbol processing space of computation in general? The latter, while conceptually interesting, seems much more difficult in terms of locating a starting point. Nature — *life-as-we-know-it* — provides numerous starting-points that can be tweaked, subverted and distorted in our search for novelty (e.g. replication, evolution, fitness, form, matter, etc.). Computation is abstract and ungrounded, inevitably needing to be made concrete through some interpretation[44]. To date, these interpretations primarily reference the metaphors on which their processes are based.

Simplistically, some of these problems can be avoided by embedding our system in physical reality (e.g. by building robots or systems that interact directly with the world through measurement and action). However, in this case, while we loose the difficulties of simulation, we do not remove the problem of predicting genuine emergence, nor do we remove the granularity of digital computation. To reverse a common truism: a real system has the same or worse difficulties of prediction as a simulation of it does.[45]

Artists and designers are always endeavouring to create works that are more than the sum of their parts. The “more” in this case is specific to the context of the artist’s concerns for the work, rather than to physical examples of emergence.

For example, Saunders defines: “Concepts derived from an existing knowledge base but which demonstrate significantly different properties are called emergent” (Saunders 1999). However, such a definition seems too broad. By this definition, an image on a television may be considered an emergent property of phosphor dots excited by electrons. This definition does not distinguish between systems in which there is no interaction between components at the micro level and those in which there are both *interaction* and *process relationships* at this level (this is the difference, for example, between a television

[44] Although one can imagine an uninteresting conceptual art work that consists of a process running on a machine with no directly perceptible output.

[45] For example, many evolutionary robotics systems spend much of their development time in simulation, as it is faster and cheaper than developing real robots.

image and a cellular automata simulation). A preferred definition is more specific about the category of phenomena it purports to distinguish.

Cariani refers to factors outside the frame of reference of the computer program in what he calls "emergence relative-to-the-model". This gives us an insight into one of the possible roles for emergence in generative art, where the "emergence" is not in the simulation itself; rather how it changes the way we think and interact with the world (Cariani 1991), and discussion in (Emmeche 1994, Chapter 6).

British artist, Richard Brown, in developing his artwork *Biotica*, "wanted creatures to spontaneously emerge from a primitive soup, rather than craft them by hand" (Brown, R. 2001). Such goals are similar for many AL artists and researchers alike who seek to develop *self-organizing* systems that lead to emergent phenomena. Self-organizing systems encode some form of physical (or pseudo- or meta- physical) relationships, including basic laws of how entities operate within the (simulated) physical system. The "trick" is in the selection of local rules that determine the nature of the resultant behaviour and the careful selection of initial conditions — this can prove illusive. By Brown's own admission, the work "did not produce any surprising emergent results." Adding complexity to the rules and simulated physical phase space of Biotica resulted in a more complex system, but not in results that created new levels of surprise, agency or novelty.

Sim's *Virtual Creatures* (Sims 1994a) on the other hand, do indeed produce novelty and surprising results, but are they truly emergent? Sims designed a specific low-level infrastructure to support his conscious goal of creating block-like creatures that discover, via competitive evolution, solutions to specific goals (following lights, competing for objects), rather than spontaneously emerging. For open-ended evolution, much more consideration needs to be given to designing the environment. For genuinely new symbolic information to arise in the genome, the entire semantically-closed organization (genotype, phenotype and the interpretation machinery that produces the latter from the former, including the whole developmental process through which an adult phenotype is produced) – needs to be "embedded in the arena of competition" (Taylor 2002).

In a design sense, it is possible to make creative systems that exhibit emergent properties beyond the designer's conscious intentions, hence creating an artefact, process, or system that is “more” than was conceived by the designer. This is not unique to computer-based design, but it offers an important glimpse into the possible usefulness of such design techniques — “letting go of control” as an alternative to the functionalist, user-centred modes of design currently popular in commercial computing (Norman & Draper 1986). Nature can be seen as a complex system that can be loosely transferred to the process of design, with the hope that human poiesis may somehow obtain the elements of physis so revered in the design world. Mimicry of natural processes with a view to emulation, while possibly sufficient for novel design, does not alone necessarily translate as effective methodology for art however.

4.6 Methodologies of Generative Art

Emergence, while part of the generative artist’s impetus, is too broad a goal in a general sense, due to a lack of reference points (unless we wish simply to mimic existing systems, creating a *simulation*). What is needed are some reference points suited to creating generative works that aspire to some genuine sense of emergence. In this section, we discuss a number of methodologies that either might be, or have been, used to produce such emergent qualities in artworks.

4.6.1 The Role of Subversion

As discussed in Section 4.5.1, computational phase space is too large to serve as a starting point alone for developing an artwork. Thus, it seems that the majority of generative art works have drawn from the palette of existing technical procedures and *subverted* them in order to expose some interesting, novel or previously unconsidered feature of such processes, or to re-interpret a particular process for the artist’s own ends (tweaking the conceptual space). For example, in the works of Australian artist John Tonkin, evolutionary processes are subverted, and used by the artist to expose political and social concepts of evolution and its implications (Tonkin 2001).

Other approaches may be to visualise, sonify, or create in unusual or context dependent ways, systems that reveal the process in hitherto unknown ways. There is no doubt that subversion and the irrational play an important role in many forms of contemporary art, seemingly much the anthesis of the standard scientific mode of practice, or that of the teleological engineer. However, generative processes in particular often appear most strongly to deny this conventional mode of thinking, providing a deeper connection with the seeming irreducibility of a strong emergent process.

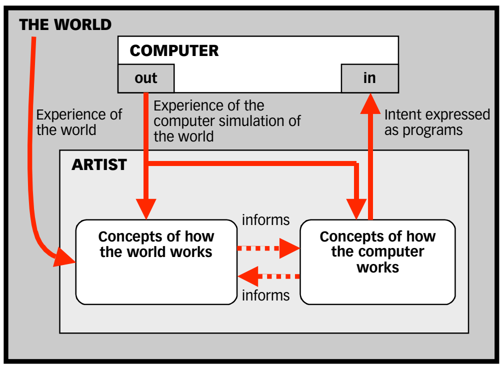
4.6.2 Symbol Manipulation, Mental Models

Consider how the generative artist might think about the relation between the world, the computer program, and its outcomes. In addition, consider too, how such relations may be different for someone using and experiencing a generative artwork (the phenotype).

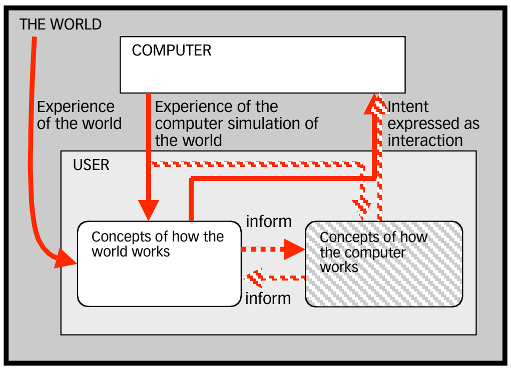
In the theory of modelling and simulation literature, a number of different *frameworks* have been proposed. These frameworks are used to help formalise the task of creating simulation from model, and model from observation. One such approach is the *System Specification Hierarchy* (SSH), which consists of a number of hierarchical levels that encapsulate different epistemological properties (Zeigler, Kim & Praehofer 2000). Specification levels include *observation frames*, *I/O function* and *behaviour*, *state transition* and *coupled component*. This framework is useful because it deals with dynamic, processed based systems that change over time. Elements of this framework will be adopted here, but it is important to stress that simulation theory has different formal goals than that of generative art, particularly where formal verification and validation are concerned.

The diagrams shown in Figure 4-1 capture aspects of a particular way of working, but one general enough to be useful. In Figure 4-1A, relationships between the generative artist, the computer, and the world, are expressed in terms of *experimental* and *observation frames* (Zeigler, Kim & Praehofer 2000, Chapter 1), *concepts* and *information flow*. Figure 4-1B shows the relationships for the user or viewer of the generative artwork. There are two key sets of concepts (models) held by the artist: *how the world works* and *how the computer works*. In order to program the computer with a view to implementing aspects of how the world works (e.g. emergent behaviour), there must be a translation or mapping between the concepts of how the world works and how the computer works. Knowledge of how the world works is informed by interactions in the world (the experimental frame). In developing models of how the world works, and implementing simulations on the computer, the computer simulation may inform us how the world works as well.[46]

As the figure shows, there are two feedback loops operating. Concepts about how the world works inform concepts about how the computer works reciprocally. Concepts of how the world works must be mediated by concepts of how the computer works before they can be implemented as a program run on the computer. The output of such a program may inform both concepts of how the world works (as this is what the simulation or artwork is attempting to achieve) and how the computer works.


THE WORLD
COMPUTER
out
in
Experience of the world
Experience of the computer simulation of the world
Intent expressed as programs
ARTIST
informs
Concepts of how the world works
Concepts of how the computer works
informs

A. *Development Mode.* Information flow for the artist (with the intent of authoring a generative artwork) interacting with the computer working in the world. The red lines show key information flows.


THE WORLD
COMPUTER
Experience of the world
Experience of the computer simulation of the world
Intent expressed as interaction
USER
inform
Concepts of how the world works
Concepts of how the computer works
inform

B. *Experience Mode.* Information flow for the user interacting with the computer, working in the world. The lighter shaded lines indicated weakened channels of information flow.

Figure 4-1: Concepts and information flow between the artist or programmer and the computer.

There are many observations we can make from these relations and feedback loops. For example, consider the mapping or translation of concepts about the world to the more limited domain of the computer. Many AL simulations map concepts about the world to the purely symbolic domain of computation

[46] Of course, we make no assumption of the accuracy of the simulation, beyond its intent to simulate some aspects of the world, thus what the computer informs us about the world may possibly be incorrect or irrelevant.

(birth, death, life), even the concept of the world itself is reduced to a finite, Cartesian, possibly discrete representation. If such representations go on to inform concepts about the (real) world, then clearly their usefulness may be limited. In simulation terms, we speak of *morphism* relationships between corresponding levels in the System Specification Hierarchy (Zeigler, Kim & Praehofer 2000, pp. 18-21). Two different observation frames are *isomorphic* if there is a one-to-one correspondence between the elements of different models (in the SSH framework, this requires respective inputs, outputs and time bases to be identical). In general, *homomorphic equivalence* is desired at the state transition level. For two systems S and S', suppose S is bigger than S' (has more states). A homomorphism is where some state transition from $s_i \rightarrow s_{i+n}, n > 1$ in S is equivalent to a single state transition in S' (i.e. $s'_k \rightarrow s'_{k+1}$) where the pairs $\left(s_i, s'_k\right)$ and $\left(s_{i+n}, s'_{k+1}\right)$ correspond to equivalent states.

Ideally, we would like an isomorphic correspondence in our simulation of observed phenomena (Suppes 1960), but this is, in practical terms, difficult or impossible for all but the simplest cases. In the semantic view, homomorphisms commonly operate between systems. For example, we would like our model to exhibit homomorphisms between simulation and reality.

Models, as they operate here, correspond to what Ronald Giere terms *representational,* where they function as tools for representing the world (Giere 1999). In a representational model, one property or state represents another property or state from the observed system. Such models are *interest relative* where the properties of the model necessarily reflect the interests of the model designer (mediated by their knowledge of how the world works). Interestingly, the semantic interpretation of model properties may be different between development mode and experience mode.

The use of frameworks and morphisms relates to the verification and validation of empirical models. These qualities are important when comparing two different models or attempting to assess critical information provided by the simulation (e.g., global warming, population dynamics). However, in the artistic context, verification and validation requires no formal proof. Frameworks do however provide a more complete basis for establishing formal specification (and accuracy) in the modelling process, regardless of application.

Consider also, how we might achieve Cariani's relative-to-the-model emergence (Cariani 1991) and how our assumptions in Figure 4-1A will be different for the user/viewer of the artwork. This is shown in Figure 4-1B. Some of the information flows are shown in a lighter shade to reflect the possibility that their effects may be diminished in the experience of the artwork. In this case, the feedback loops become minimal or disappear, meaning we may have to work harder to achieve emergent behaviour. One important way this can be obtained is by strengthening the connection between the user's interaction with the model and their concepts of how the world works. To achieve relative-to-the-model emergence, engagement with the computer needs to suggest that the work is *more* than its design intended it to be — it must be *informationally open*. For artworks, this might be achieved in a number of ways:

- through interaction (a feedback loop) with the work in real-time, where continuous re-assessment of the work suggests (for example) dynamics beyond the physical or virtual elements that compose the work;
- through suggestion of the sublime by an apparent vastness — that the simulation's representation of the world is broader than the user's concept matching[47] of the same phenomena. This is the subject of discussion in the next section.

A more problematic area is that of the user's concepts about how the computer works, what it is capable of, and so forth. Naively, people may find some things the computer does impressive because the computer is doing them (rather than a person for instance). We are fascinated and amazed in many cases, that a mere machine can produce things of seemingly un-machine-like qualities — technical prowess, even qualities we only associate with, for instance, nature itself. Of course, such assumptions reveal gaps in our analysis of both the world and the machine. These gaps can be easily exploited; making computer-based works seems more impressive than they really are.

[47] As used here "concept matching" means matching the experience of some phenomena to concepts held by the person experiencing those phenomena.

4.7 The Computational Sublime

The sublime has a long and well-explored history in art, particularly in the eighteenth and nineteenth centuries. Kant distinguished the *mathematically sublime* and the *dynamically sublime*. The mathematically sublime brings to our attention that which we *can* conceive of symbolically (through mathematics), but *cannot* experience sensorially. The dynamically sublime suggests the incomprehensible power of nature. An important aspect of the sublime is the tension created between pleasure and fear: the pleasure of knowing that we can be aware of what we cannot experience and the fear that there exist things that are too vast or powerful for us to experience. In relating nature and aesthetics, the sublime formed a major critical and philosophical approach in western art in the eighteenth and nineteenth centuries (for an overview of the relation of the sublime to nature see (Soper 1995, Chapter 7)). Bourke and Kant argued that it is possible to suggest the sublimity of nature (that which we cannot experience in totality) through the experience of an artwork.

In recent times, the *postmodern sublime* has contrasted *beauty* as a form that can be comprehended against the *sublime,* as that which is unrepresentable in sensation (Lyotard 1984). As discussed, emergence in computation is unrepresentable, in the sense that the product of elements interacting in ways gives rise to properties that cannot be predicted. Artworks that seek to give a sense of the processes of nature in machines, seek to give experience to that which cannot be experienced in totality — only suggested through a dynamic interaction.

Therefore, the concept of the *computational sublime* is introduced: the instilling of simultaneous feelings of pleasure and fear in the viewer of a process realised in a computing machine. A duality, in that even though we cannot comprehend the process directly, we can experience it through the machine — hence we are forced to relinquish control. It is possible to realise processes of this kind in the computer due to the speed and scale of its internal mechanism, and because its operations occur at a rate and in a space vastly different to the realm of our direct perceptual experience.

An example of a work that subverts standard technological processes and suggests the role of the computational sublime is that of the Dutch artists Erwin Driessens & Maria Verstappen (Driessens & Verstappen 2001a). Their work, *IMA Traveller* subverts the traditional concept of cellular automata by making the automata recursive, leading to qualitatively different results to those achieved through direct mimicry of technical CA techniques in other generative works. *IMA Traveller* suggests the computational sublime because it is in effect, an infinite space. It offers both pleasure and fear: pleasure in the sense that here inside a finite space is the representation (and partial experience) of something infinite to be explored at will; fear in that the work *is* in fact infinite, and also in that we have lost control over it. The interaction is somewhat illusory, in the sense that while we can control the zoom into particular sections of the image, we cannot stop ourselves from continually falling (zooming) into the work, and we can never return to a previous location in the journey. The work creates an illusion of space that is being constantly created for the moment (as opposed to works that draw from pre-computed choice-sets). The zooming process will *never* stop. That there is no real ground plane or point of reference suggests Kierkegaard's quote of Section 4.4.3: you are always going, but only from the point of where you've been.

4.8 Conclusions

Generative processes offer a rich and complex area for artists to explore. This chapter has only touched upon a few issues, and ignored many of importance. The concept of emergence, though constantly changing and often criticised, is a recurring philosophical theme that evolved to supersede the vitalist philosophies of the nineteenth century. This philosophical theme links a number of schools of thought in the sciences of the twentieth century — systems theory, cybernetics, and artificial life. If such a theme could be expressed succinctly, it would be as a *philosophy of process* that includes both mechanism and matter as fundamental properties of the universe.

It is important however, to consider the fate and usefulness of both systems theory and cybernetics. Systems theory was a cultural reaction to reductionism and highly specific modes and languages in science. Cybernetics (as defined by Ashby) was a theory of machines, but it treats, not things, but ways of behaving (Ashby 1952). Today, at the beginning of the twenty-first century, while both these disciplines still have their proponents, such a holistic approach is not the predominate methodology for training scientists.

The longevity of artificial life is yet to be determined. As the stepchild of systems theory and cybernetics, AL, once again, is hedging its bets on the process philosophy. The real payoffs and long-term goals of AL, such as the creation of artificial systems that we can confidently call *alive*, have yet to materialise. AL may not have the same escape hatches as AI did, but even by failing to create our own artificial living systems, we may still be learning about life.

Generative art draws from the philosophies of the process-based sciences. Its potential is rich: art-as-it-could-be, artworks that are autonomous, genuinely novel, emergent, active, self-renewing and never-ending. Yet, while many innovative generative artworks have given us glimpses into these possibilities, such lofty goals remain intangible at present, and with no guarantee of success. It remains for future generations of generative artists to determine if any of these goals will be achieved.

This chapter has discussed some modes and methodologies for generative art to explore: the role of subversion; mental models of understanding for the artist and audience; the computational sublime. I do not suggest that these are the only issues for consideration, and inevitably, artworks will be judged not only on the themes explored here, but also in terms of the more comfortable and fashionable theories of the electronic and new-media arts, and contemporary art in general.

Generative art seeks to exploit the out-of-control nature of nature, but to achieve this in a genuine sense the artist is obliged to acknowledge that control must really be relinquished. This abandonment remains a very difficult thing to achieve, and a challenge to the conceptual processes of developing an artwork.

I have also acknowledged that computation, as it exists today, may never be able to give us true emergence, the likes of which we observe in the world around us. As a number of authors have shown, the dialectic of simulation and realisation, of life-*like* and life, are fundamental issues still seeking a stronger resolution.

Such a prognostication views modern sculpture as a preparatory stage representing steps towards the simulation of biological life, a point in human evolution when the sculptor begins to imitate the machine maker and the creator of scientific models, unaware that the artifacts of technology are meant to do the same things as his own forms, and that they do them more successfully. … The machine, he intuitively realized, as unstable and inefficient as it was, remained the only means by which man would eventually reconstruct intelligent life, or what might be called life-bearing artifacts. As a result, much modern sculpture has been concerned with the creation of pseudo-machines which haphazardly approximate the life impulse. (Burnham 1968a)

Part 2

5 Generative Modelling with L-systems

To see a world in a grain of sand
And a heaven in a wild flower,
Hold infinity in the palm of your hand
And eternity in an hour.
— *William Blake, ca. 1803*

The previous chapters provided overviews of salient issues related to certain types of art practice, such as the relation of science to art, the fields of Artificial Life and generative art, issues of emergence, levels and patterns, and the computational sublime. I now turn to a technical description of my research into generative systems for artistic purposes. Here I provide a survey of related work in computer graphics, organic and botanical modelling and then go on to describe the specific systems developed in detail.

In particular, this chapter looks at generative modelling using string rewriting grammars (L-Systems) formalisms originally conceived as a theoretical framework for the developmental modelling of multicellular structures, such as plants.

5.1 Modelling Taxonomies

In the 1980s, a number of computer graphics researchers turned their attention to the "realistic" modelling of "natural phenomena" for the purposes of dynamic visualisation of natural forms and processes. Common examples in computer graphics include: shells (Fowler, Meinhardt & Prusinkiewicz 1992); plants (Prusinkiewicz & Lindenmayer 1990); trees and flowers (de Reffye et al. 1988); ocean waves (Fournier & Reeves 1986; Mastin, Watterberg & Mareda 1987; Peachey 1986); ocean and terrain (Max 1981), fractal terrain (Fournier, Fussell & Carpenter 1982); clouds (Gardner 1984, 1985); waterfalls (Sims 1990b); fire (Gardner 1990; Stam & Fiume 1995); natural "textures" such as stone and marble (Peachey 1985; Perlin 1985b), "organic" cellular textures (Fleischer et al. 1995); and animal skin patterns (Turk 1991; Witkin & Kass 1991).

Interestingly, "natural phenomena" rarely included complete animals. When attempts were made to model animals, the modelling was limited to particular facets, such as the motion dynamics of snakes and worms (Miller, G. S. P. 1988), or the group behavioural characteristics of flocks, herds and schools (Reynolds 1987). Modelling of individual animals in detail involves a number of separate sub-topics in computer graphics, such as inverse kinematics, articulated body dynamics, complex surfaces and texturing. Each considered somewhat separate problems for a dynamic visual simulation.[48]

Fournier proposed a taxonomy of different approaches to the modelling of natural phenomenon (Fournier 1987).

- *Empirical models* — data generated from empirical sources, such as terrain models acquired from digitisation systems, *geographic information systems* (GIS), or models constructed as a result of direct observation and measurement.
- *Physical models* — models generated by simulating domain specific physical laws, such as the wave simulation of Miyata (Miyata 1986), or the cloud models of Kajiya and Von Herzen (Kajiya & Von Herzen 1984). Physical modelling is a more general topic in simulation and computer graphics, including areas such as rigid and soft-body dynamics (Barzel 1992).
- *Morphological models* — models that directly give the shape of the model, for example the parametric shell models of D'Arcy Thompson (Thompson 1961). In a wider context, these models are often know as

[48] Developments in the following decade saw attempts to incorporate all these elements. See for example: (Tu & Terzopoulos 1994).

functional, as they are defined by some closed-form mathematical function.

- *Structural models* — models defined by formal specification of structure, with the interpretation of that structure a separate component of the model. This includes the formal language approach of Smith (Smith 1984) and the string re-writing grammars developed by Lindenmayer (Lindenmayer 1968; Lindenmayer & Rozenberg 1976). In the context of plant modelling, Hanan calls these *architectural* and further categorises them into *developmental* and *non-developmental* models (Hanan 1992).
- *Impressionistic models* — do not seek any structural or morphogenic realism, rather a "softer" impressionistic style in their visualisation, similar to that of the impressionist painters. Examples include the cloud and tree models of Gardner (Gardner 1984, 1985) and forest and tree models of Reeves and Blau (Reeves & Blau 1985).
- *Self-models* — are self-defined models such as Julia and Mandelbrot sets (Mandelbrot 1983; Peitgen & Richter 1986; Peitgen & Saupe 1988). In a mathematical sense, these may also be classified as functional models.
- *Mixed models* — models of mixed type that do not easily fit into any of the above categories, or combine elements of the other categories.

It can be argued that most methods used to model natural phenomena fall into the more widely used *physically based*, *functionally based* and *procedurally based* categories common in computer graphics classifications (Barr et al. 1988; Ebert et al. 1993; Ebert et al. 1994; Inakage et al. 1988). In classifications used in botanical modelling, Waller and Steingraeber define *spatial* models as those providing topological and geometric information necessary to produce an image (Waller & Steingraeber 1985). Non-spatial models focus on global characteristics often providing statistical information rather than specific geometric datasets. Within the classification framework listed above, all models would be described as spatial.

5.2 Generative Modelling

In general, computer graphic models of natural scenes reflect the complexity of nature and therefore the data they generate can be relatively large.

Models that define some *process* (be it physical, functional or procedural), and which generate data from that process (which is the primary output of the model), are known as *generative models*. In the modelling of natural scenes, generative models often use the principle of *database amplification* (Smith 1984), where the model is many orders of magnitude smaller than the data it generates. This may often be an advantageous approach, since manipulating large datasets directly can be difficult, whereas the management of a far simpler generative specification is potentially easier. The principle of database amplification is often seen in nature, for example in the expression of DNA into an organism.

The modelling systems described in this thesis falls into the category of *developmental procedural*, whereby development of geometric datasets is controlled by *endogenous* information flow. What makes this system different from similar systems is both its preservation of temporal continuity and its ability to allow natural hierarchies in procedural specification.

5.2.1 Modeller–Renderer Relationships

Generation of large geometric datasets may present visualisation problems due to the time and memory requirements needed to process the geometric data. In this situation, designing a system with some form of "intelligent" on-demand processing can be advantageous. Hart classifies procedural generative methods into "data amplifier" or "lazy evaluation" paradigms (Hart 1996). These criteria relate to the relationship between geometric data generation and the rendering of that data. With the data amplifier paradigm, geometric data for the entire scene is generated by the modeller and sent to the rendering system to be turned into raster images. With lazy evaluation, the renderer drives the process calling for geometric detail where needed, dependent on visibility and level-of-detail required.

Natural scenes often contain repeated elements over a number of hierarchies; these elements can be either geometrically identical or statistically similar. Ultimately, geometric specification must be decomposed into a set of geometric primitives that the renderer is capable of rendering natively (i.e. without further decomposition), typically triangles or convex polygons in hardware-based systems (Woo et al. 1997). In software rendering systems this may vary, for example the *RenderMan* system decomposes all geometry into sub-pixel, *micropolygons* (Cook, R. L., Carpenter & Catmull 1987). Ray-tracing based systems allow the direct rendering of more complex primitives such as algebraic surfaces, quadric surfaces, parametric surfaces, volumetric data, “soft” objects, as well as polygons (Hanrahan 1983; Heckbert 1987; Kajiya 1983; Kajiya & Von Herzen 1984; Levoy 1990; Whitted 1980).

The statistical similarity of many higher order structures may result from the transformation and organization of simpler geometric entities. A scheme that can capture these hierarchical relationships can then take advantage of a more efficient relationship between model generation and renderer.

For example, consider a field of flowers of the same species — each flower may be statistically similar, each of the elements that comprise individual flowers may be topologically identical (petals, florets, stamen, etc. made from individual geometric primitives — see Section 7.3.1). In this situation, *instancing* of geometric data at a number of levels provides an efficient method of minimizing the volume of data-flow between the modeller and renderer.

5.2.2 Developmental Models

Developmental models are often used in the biological sciences. They capture the developmental and morphogenic changes an organism or population undergoes in its lifetime. In the sense that the term is used in this thesis, developmental models imply some form of *time-based* modelling, i.e. that temporal specification is an integral part of the model. A developmental model, suitable for animating morphogenic forms, is presented in Chapter 9.

5.3 L-systems

Lindenmayer systems, or L-systems were originally conceived as a theoretical modelling framework for the development of multi-cellular organisms, such as plants (Lindenmayer 1968). The original emphasis was on plant topology

— neighbourhood relations between plant cells or higher structures. They arose from an interest in string rewriting based on Chomsky's work on formal grammars in the 1950s (Chomsky 1956). The main difference between Chomsky grammars and L-systems is that with Chomsky grammars, productions are applied *sequentially* as opposed to L-systems where productions are applied in *parallel*.

Initially, results were interpreted by looking at the words generated during a derivation. Simple graphical substitution for individual letters was applied, resulting in schematic images. With the advent of computer graphics displays capable of displaying colour images and manipulating two- and three-dimensional primitives, researchers looked at more advanced methods of visualizing produced strings. These attempts are summarised in the following section.

5.3.1 Visualisation of Produced Strings

In order to visualise detailed botanical and biological structures, particularly the branching structures found in plants, more complex interpretation techniques were required. Aono and Kunnii were the first to consider the use of L-systems for generating realistic botanical images (Aono & Kunii 1984). However, they found the original DOL-system[49] "not powerful enough" to represent complex branching structures. They added additional drawing rules in addition to the L-system to generate three-dimensional tree models exhibiting monopodial and dichotomous branching patterns. Hogeweg and Hesper studied *propagating*, *deterministic* bracketed 2L-Systems to produce a rich variety of tree structures, but their graphical results were limited to 2D black and white line drawings (Hogeweg & Hesper 1974). Similarly, Frijters and Lindenmayer made simple graphic interpretations, focusing on topology with geometric enhancements made in a post-processing step (Frijters & Lindenmayer 1974).

The use of L-Systems for computer graphics modelling was developed later by Smith (Smith 1984) who coined the term *graftals* and referred to the "fractal" nature of structures that can be formed by rewriting grammars. Influenced by the use of interpreting individual letters with plotter pen com-

[49] Deterministic, context-free L-system, defined in section 5.3.2.1.

mands (Szilard & Quinton 1979), Prusinkiewicz used a LOGO-style *turtle interpretation* (Abelson & DiSessa 1982) to create two- and three-dimensional models from the strings produced by L-systems (Prusinkiewicz 1986b, 1987). He showed the technique is particularly useful in modelling herbaceous (non-woody) species. Highly detailed and realistic images were created by extending grammars, providing pre-defined surfaces for plant elements that were difficult to model within the L-system directly, adding advanced turtle interpretations, and rendering models using a ray tracing technique (Prusinkiewicz & Hanan 1989; Prusinkiewicz & Lindenmayer 1990; Prusinkiewicz, Lindenmayer & Hanan 1988).

The issue of symbol and turtle interpretation will be discussed in more detail in Chapter 6.

5.3.2 0L-systems

0L-systems are the simplest class of L-system, being *context-free, interactionless,* or *zero-sided.* Following the terminology and presentations found in (Hanan 1992; Herman & Rozenberg 1975; Prusinkiewicz & Lindenmayer 1990; Rozenberg & Salomaa 1980), a formal description is presented below.

5.3.2.1 Definition of 0L-systems

A context-free, 0L-System is defined as the ordered triple $G = \langle V, \omega, P \rangle$ where:

- $V = \{s_1, s_2, \ldots, s_n\}$ is an alphabet composed of a set of distinct *symbols,* s_i;
- $\omega \in V^+$, a non-empty word (sequence of symbols) over V is known as the *axiom*;
- $P \subset V \times V^*$ an endomorphism defined on V^*, known as the finite set of productions.

A production $(s, \chi) \in P$ is written in the form $s \to \chi$, where the symbol s is known as the *predecessor* and the word $\chi \in V^*$ the *successor* of the production. Where there is no specific production for a symbol s, the *identity production* $s \to s$ is assumed.

Deterministic L-systems *(DOL-systems)* have at most one production for each symbol. A 0L-system is *deterministic* if and only if for each $s \in V$ there is exactly one $\chi \in V^*$ such that $s \rightarrow \chi$.

The L-system, *G* applies a production to a symbol, *s,* in a word when the predecessor for that production matches *s*. The notation $s \mapsto \chi$ means module *s* produces a word χ as a result of applying a production in *G*.

Let $\mu = s_1 s_2 ... s_k$ be an arbitrary word over V. The word $\nu = \chi_1 \chi_2 ... \chi_m$ is generated by *G* from μ, denoted $\mu \Rightarrow \nu$, if and only if $s_i \mapsto \chi_i \;\; \forall i : i = 1, 2, \ldots, k$. *G generates* a *developmental sequence* of words $\mu_0 \mu_1 ... \mu_n$ by applying the matching productions from *P* at each iteration, beginning with the axiom, ω. That is $\mu_0 = \omega$, and $\mu_0 \Rightarrow \mu_1 \Rightarrow ... \Rightarrow \mu_n$. A word ν is a *derivation of length n* if there exists a developmental sequence such that $\nu = \mu_n$.

For example, this DOL-system

$$\begin{aligned} & V = \{F, R, L, [,]\} \\ & \omega : F \\ & p_1 : F \rightarrow FFR[RFLFLF]L[LFRFRF] \end{aligned} \tag{5.1}$$

Generates the sequence of words:

$$\begin{aligned} \omega = \mu_0 = \;\; & F \\ \mu_1 = \;\; & FFR[RFLFLF]L[LFRFRF] \\ \mu_2 = \;\; & FFR[RFLFLF]L[LFRFRF]FFR[RFLFLF]L[LFRFRF]R \\ & [RFFR[RFLFLF]L[LFRFRF]LFFR[RFLFLF]L[LFRFRF] \\ & LFFR[RFLFLF]L[LFRFRF]]L[LFFR[RFLFLF]L[LFRFRF] \\ & RFFR[RFLFLF]L[LFRFRF]RFFR[RFLFLF]L[LFRFRF]] \end{aligned}$$

As can be seen, the size of the string increases rapidly under such a production, illustrating the generative nature of even simple productions. Moreover, it is easy to see recursive patterns developing in the produced strings, leading to self-similarity in the visualisations of the string.

5.3.2.2 A Note on Symbol Notation

Traditionally, the symbols s_i of the alphabet V are notated using single alphabetic characters or special characters such as "[", "]", "+", "–", "&", "^", "|", "\", "/", and so on. Throughout the rest of this thesis, symbols with multi-character representations are also used. This mirrors the actual computer implementation, where identifiers denoting specific symbols can be composed of multiple alphanumeric characters, but must always begin with an alphabetic character: [a-zA-Z].

5.3.3 Context Sensitive L-systems

Context sensitive L-systems use contextual relations to determine the appropriate production. A wide variety of different types of context sensitive L-systems have been detailed in the literature (Herman & Rozenberg 1975; Lindenmayer 1968; Lindenmayer & Rozenberg 1976; Salomaa 1973). Since the main concern here is in the practical application of formalisms to the electronic arts, rather than a contribution to the theory of formal languages, a simplified and pragmatic description will be given here. This follows the notation and definitions of (Hanan 1992; Prusinkiewicz & Lindenmayer 1990).

A particular generalisation of context sensitive L-systems, known as *(m,n)L-systems* will be considered. In this case, m and n refer to the number of symbols on the left and right sides respectively of the predecessor symbol under consideration. These additional symbols will be considered in determining a match for the predecessor symbol of a given production. In specifying such a production, the following notation is used:

$$s_{l_1}s_{l_2}\ldots s_{l_m} < s > s_{r_1}s_{r_2}\ldots s_{r_n} \rightarrow \chi$$

where the symbol s (known as the *strict predecessor*), can produce the word sequence χ if and only if the sequence $s_{l_1}s_{l_2}\ldots s_{l_m} s\, s_{r_1}s_{r_2}\ldots s_{r_n}$ exists in the developmental word μ_i of the current iteration. The values of m and n may be different for each production in the set of productions for a given L-system. It is also possible for m or n to be 0. If $m = n = 0$, then the production is no longer context sensitive.

The sequence $s_{l_1}s_{l_2}\ldots s_{l_m}$ is known as the *left context* of *length m*, $s_{r_1}s_{r_2}\ldots s_{r_n}$ the *right context* of *length n*, of s in a production p_i.

An *(m,n)L-system* is deterministic if and only if no two productions can match the same symbol in a string. Since there are no restrictions on the lengths of context it is quite easy to devise *(m,n)*L-systems that are non-deterministic, hence some resolution rule is required to determine which particular production should be applied when more than one matches. Here, a number of possible approaches have been considered:

1. Use the first matching production (this implies productions must be *ordered*);
2. Select the production randomly from the set of matching productions;
3. Use the matching production with the highest (most specific) context, i.e. the matching production with the highest value of $m+n$. If more than one production has this highest value then select the first production from this set of highest value matching productions;
4. Same as 3, except that in the case of multiple highest value contexts, randomly select from the set of highest value matching productions.

Note that 3 can be achieved using 1 if productions are ordered in descending length of context (similarly for 2 and 4). As an example, consider the following *(m,n)*L-system:

$$\begin{aligned} &\omega : ABAB \\ &p_1 : A > BAB \rightarrow C \\ &p_2 : A > BA \rightarrow D \\ &p_3 : A > B \rightarrow E \end{aligned} \tag{5.2}$$

In this case, all three productions match the first symbol of the axiom. Applying rule 1 or 3 to resolve this ambiguity would result in p_1 being applied. If the order of rules were changed however:

$$\begin{aligned} &\omega : ABAB \\ &p_1 : A > B \rightarrow E \\ &p_2 : A > BA \rightarrow D \\ &p_3 : A > BAB \rightarrow C \end{aligned} \tag{5.3}$$

Rule 1 would apply p_1, rule 3 would apply p_3.

The choice of appropriate resolution depends on the application. Using rule 1 above is used in most cases (Prusinkiewicz & Lindenmayer 1990), rule 2 defines a non-deterministic L-system (see next section, 5.3.4). Choosing the productions with higher context sensitivity may be useful to allow more specific rules to be used in specialised circumstances, while more general situations utilise productions with contexts of smaller lengths (this is analogous to the rule-based learning systems described in (Holland 1995), whereby specific rules are preferred over general ones). It also removes the ordering constraint that may make large production sets more readable, hence easier to construct and change.

5.3.4 Stochastic 0L-systems

With the exception of the use of rule 2 in the preceding section, only deterministic L-systems have been described. For a given L-system, the application of productions is deterministic; hence, the models generated by the same L-system will be identical. However, we may wish to introduce some variation in the way productions are applied allowing variation in both topology and geometry. Stochastic L-systems will also find application in simulating Markov models for music in Section 11.3.4.

Here the definition of stochastic L-systems is from (Prusinkiewicz & Lindenmayer 1990, Section 1.7), which is similar to that of (Eichhorst & Savitch 1980; Yokomori 1980).

5.3.4.1 Definition of Stochastic 0L-systems

A *stochastic 0L-system* is an ordered quadruplet $G_\pi = \langle V, \omega, P, \pi \rangle$, where:

- the alphabet, V, axiom ω and set of productions, P are as defined for deterministic 0L-systems in Section 5.3.2.
- The function $\pi : P \to (0,1]$ is called the probability distribution and maps the set of productions to a set of production probabilities.

Let $\hat{P}(s) \subset P$ be the subset of productions with s as the predecessor. If no production for s is specified the identity production, $s \to s$ is assumed. Each production $p_i \in \hat{P}(s)$ has with it an associated probability $\pi(p_i)$, where

$$\sum_{p_i \in \hat{P}(s)} \pi(p_i) = 1 \tag{5.4}$$

The derivation $\mu \Rightarrow v$ is known as a *stochastic derivation* in G_π if for each occurrence of s in the word μ, the probability of applying production p_i is equal to $\pi(p_i)$. In a single word, different productions with the same predecessor may be applied in a single derivation step. Selection is weighted according to the probabilities associated with each production with the appropriate predecessor.

A production in a stochastic L-system is notated $p_i : s \xrightarrow{\pi(p_i)} \chi$.

5.3.5 Parametric 0L-systems

Parametric L-systems were proposed to address a number of shortcomings in previous L-system models, particularly for the realistic modelling of plants. As string re-writing systems, L-systems are fundamentally discrete. This discrete nature makes it difficult to model many continuous phenomena or accurately represent irrational ratios.

Lindenmayer recognised these difficulties and proposed the association of numerical parameters with symbols (Lindenmayer 1974). With application to both string rewriting and turtle interpretation, Prusinkiewicz and Hanan described *parametric L-systems*, whereby a group of continuous parameters are associated with L-system symbols and used in the turtle interpretation of those symbols (Prusinkiewicz & Hanan 1990). The application of parametric L-systems to plant modelling was further developed by Hanan in his Ph.D. thesis (Hanan 1992). The definition below is based on the definitions found in those references.

5.3.5.1 Definition of Parametric 0L-systems

Parametric L-systems associate real-valued *parameters* with each symbol, collectively forming a *parametric module*. A module with symbol $S \in V$ and parameters $a_1, a_2, ..., a_n \in \Re$ is written $S(a_1, a_2, ..., a_n)$. Strings of parametric modules form *parametric words*. It is important to differentiate the real-valued *actual* parameters of modules, from the *formal* parameters specified in productions. In practice, formal parameters are given unique[50] identifier names when specifying productions.

Assuming the following definitions:

- Σ is the set of formal parameters, $C(\Sigma)$ is a logical expression using parameters from Σ, $E(\Sigma)$ is an arithmetic expression with parameters from the same set.

[50] Within the scope of the associated production.

- C and E consist of formal parameters and numeric constants, combined using the standard operators +, –, /, *, ^ (exponentiation) $\sqrt[n]{}$ (nth root, defaulting to n=2 if n is not specified); relational operators $<,>,\leq,=,\neq$; logical operators ! (not), & (and), | (or); a number of trigonometric, stochastic and other functions, detailed in Section 6.8; and parentheses “(“, “)”. Rules for constructing expressions, operator precedence and associativity are the same as for the C programming language (Kernighan & Ritchie 1988).
- $\mathcal{C}(\Sigma)$ and $\mathcal{E}(\Sigma)$ are the sets of correctly constructed logical and arithmetic expressions with parameters from Σ. Logical expressions evaluate to Boolean values of TRUE or FALSE (equivalent to 1 or 0). Logical expressions evaluate to a real number in an arithmetic context.

A *parametric 0L-system* is defined as an ordered quadruplet $G = \langle V, \Sigma, \omega, P \rangle$ where:

- $V = \{s_1, s_2, \ldots, s_n\}$ is an alphabet composed of a set of distinct *symbols,* s_i;
- Σ the set of formal parameters;
- $\omega \in \left(V \times \Re^*\right)^+$, a non-empty parametric word known as the axiom; and
- $P \subset \left(V \times \Sigma^*\right) \times \mathcal{C}(\Sigma) \times \left(V \times \mathcal{E}(\Sigma)^*\right)^*$ the finite set of productions.

Productions $(\underline{s}, C, \underline{\chi})$ are denoted $\underline{s} : C \rightarrow \underline{\chi}$ where the formal module $\underline{s} \in V \times \Sigma^*$ is the *predecessor*, the logical expression $C \in \mathcal{C}(\Sigma)$ is the *condition* and $\chi \in \left(V \times \mathcal{E}(\Sigma)^*\right)^*$ is a formal parametric word known as the *successor.* A formal parameter appears once in the predecessor. The number of formal parameters must be consistent for any given symbol and match the number of actual parameters for the same symbol. A production without a condition has an implicit condition constant value of TRUE.

A production is applied to a module in a parametric word if the production *matches* that module. The necessary conditions for matching are: if the module and production predecessor symbols *and* parameter counts match; the condition statement, *C,* evaluates to TRUE when the module’s actual parameters are bound to the formal parameters as specified in the predecessor module. When a module is matched it is replaced by the successor word, χ, whose formal parameters are evaluated and bound to the corresponding actual parameters.

5.3.5.2 Derivation and Developmental Sequence Generation

The parametric L-system, G applies a production to a module m in a parametric word when the production matches m. The notation $m \mapsto \chi$ means module m produces a parametric word χ as a result of applying a production in G.

Let $\mu = m_1 m_2 ... m_j$ be an arbitrary parametric word over V. The word $\nu = \chi_1 \chi_2 ... \chi_k$ is *generated* by μ, denoted $\mu \Rightarrow \nu$, if and only if $m_i \rightarrow \chi_i$ $\forall i : i = 1, 2, \ldots, j$. G generates a *developmental sequence* of words $\mu_0 \mu_1 ... \mu_n$ by applying the matching productions from P at each iteration, beginning with the axiom, ω. That is $\mu_0 = \omega$, and $\mu_0 \Rightarrow \mu_1 \Rightarrow ... \Rightarrow \mu_n$. A word ν is a *derivation of length n* if there exists a developmental sequence such that $\nu = \mu_n$.

For example, the following parametric L-system

$$\begin{aligned} \omega &: A(1,1) \\ p_1 &: A(x,y) \rightarrow A(y, y+x) \end{aligned} \tag{5.5}$$

generates the derivation sequence

$$A(1,1) \Rightarrow A(1,2) \Rightarrow A(2,3) \Rightarrow A(3,5) \ldots \tag{5.6}$$

calculating the Fibonacci sequence. An example with a condition

$$\begin{aligned} \omega &: A(1,2) \\ p_1 &: A(x,c) : x * x \neq c \rightarrow A\left(\frac{1}{2}\left(x + \frac{c}{x}\right), c\right) \end{aligned} \tag{5.7}$$

computes the square root of the second parameter of A (labelled c) using Newton's method:

$$A(1,2) \Rightarrow A(1.5,2) \Rightarrow A(1.416667,2) \Rightarrow A(1.414216,2) \Rightarrow \ldots \tag{5.8}$$

For parametric L-systems to be *deterministic* no two productions can match the same module in a derivation word by the definition above. However, ensuring determinism by these criteria can be difficult to prove for all possible combinations of parameter values, hence a practical solution is to *order* the set of productions and apply the first production that matches in the list. If there are no matching productions, the identity production is assumed and the parameters for that module remain unchanged.

6 Turtle Interpretation

FFF[+FF[+F][-F]-FF[+F][-F]]!

— *Turtle string.*

The *turtle interpretation* of produced strings from L-systems has become a popular method for generating visual models of plants. The term "turtle interpretation" is somewhat of a misnomer, based on the concept of LOGO turtle geometry (Abelson & DiSessa 1982) where a mechanical turtle holding a pen responds to simple user commands to more around a two-dimensional surface (e.g. "turn left", "turn right", "move forward x units"). In addition, pen control commands ("pen up", "pen down", "change colour") permit drawing to be carried out, either on real paper in the case of a mechanical turtle or as a drawing on the computer screen in the case of a simulation.

In the turtle interpretation of L-systems, a derivation word is read sequentially from left to right, with certain symbols having an interpretation for the turtle (others are ignored). The turtle, moving in three-dimensional space, interprets these commands to create geometric structures. These geometric structures can then be visualised using standard computer graphics techniques. The earliest interpretations involved line drawings with a simple repertoire of turtle commands. As more detailed and realistic models were required, more complex turtle commands were added. In the system developed in this thesis, commands build, effect, and control a *scene graph*. The scene graph is a dynamic structure for representing three-dimensional geometry (Döllner & Hinrichs 1997).

6.1 Specification and Dataflow

A complete system to generate visual models based on L-systems requires more information than just the L-system itself. In the system described here, a number of predefined variables specify default behaviours and parameters that are used by the system in the absence of any other over-riding information. These include the default length, angles, and derivation length (herein referred to as l, δ, and n_μ respectively) used in turtle interpretation of produced strings.

The basic dataflow model representing a system that implements the formalisms defined in the preceding chapter is outlined in Figure 6-1. A file containing L-system definitions[51] is read into the system and geometric datasets are produced. These datasets may be visualised in real-time using appropriate graphics hardware, or sent to a software rendering system.

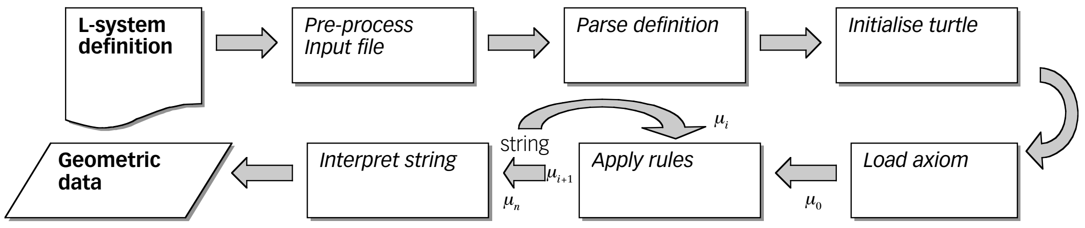


Figure 6-1: Basic dataflow for generating visual models from L-systems. Rule application is an iterative process, proceeding until the required derivation length is reached.

The L-system definition file is first pre-processed by a system equivalent to the macro processor, *cpp* (C pre-processor) (Kernighan & Ritchie 1988, pages 228-233). This permits macro replacements of, for example, parameter constants and functions, making the L-system easier to understand and modify. Comments, enclosed by the characters "/*" and "*/" or beginning with "//" and terminated by an end of line, are also removed by the macro processor.

In addition to the L-system definition itself, the input file may include a number of compiler *directives* and *statements*. Directives control symbol interpretation, read predefined surfaces and control geometric output. Global turtle parameters, such as the l and δ constants may be assigned new values.

[51] Axioms, productions, control statements and ancillary definitions.

6.2 Symbol Equivalence

L-systems are fundamentally symbol re-writing systems and interpretations are applied to some of the symbols produced by the re-writing process. Situations arise where we would like to have two or more *different* symbols that have the *same* interpretation (i.e., perform the same function when interpreted). This is particularly true in a turtle interpretation where symbols are interpreted as turtle commands. For example the "+" symbol represents a turn about the $\vec{\mathbf{U}}$ vector (see Section 6.3). We might want a different symbol, "`turn`" which has the same function as "+" but different re-writing rules.

To achieve this functionality, the *equivalent* directive is used. The syntax for use is as follows:

equiv baseSymbol equivalentSymbols …

So, for example:

```
equiv + turn plus l
```

Defines three new symbols "turn", "plus" and "l" that are interpreted by the turtle as being equivalent in function to that of the "+" symbol (in the actual computer implementation statements are terminated by a semicolon character: ";"). In the case of parametric symbols, each symbol will have its own unique set of parameters. The usefulness of the `equiv` directive will become more apparent in the chapter on developmental models and animation (Chapters 8 and 9).

6.3 Turtle state

The turtle interpreting the L-system string maintains a set of attributes, known as a *state*, which includes the following:

- Turtle *position* in Cartesian co-ordinates, denoted $\mathbf{t} = \left(t_x, t_y, t_z\right)$, relative to a world co-ordinate system, $\mathbf{C}_W$ (refer Figure 6-3).
- Turtle *orientation* in space, represented by the orthogonal unit vector triplet $[\vec{\mathbf{H}} \quad \vec{\mathbf{L}} \quad \vec{\mathbf{U}}]$, corresponding to *heading*, *left* and *up* directions respectively. Each vector is of unit length and satisfies the equation

$\vec{\mathbf{H}} \times \vec{\mathbf{L}} = \vec{\mathbf{U}}$. The combined turtle position and orientation thus form a co-ordinate system, $\mathbf{C}_\mathrm{T}$ (refer Figure 6-3).

- A generalised homogeneous *transform*, $\mathbf{T}_\mathrm{T}$ represented as a 4 x 4 matrix. Turtle commands may change this matrix, which is then pre-multiplied with instanced geometry (detailed in Section 6.3.1). This transform can be used for basic geometric transformations such as scaling or skewing.
- The current drawing material, a context dependent value determined by the mode of geometric output. In the case of simple vector based drawing it is simply a colour, represented by a normalised scalar RGB triplet (c_R, c_G, c_B) whose components specify red, green and blue colour values. For more complex geometric output, the current drawing material consists of more detailed material descriptions[52], which include properties such as ambient, diffuse and specular reflections, transparency, texture and bump mapping parameters.
- Current drawing parameters such as *line width* (which specifies the width (line) or radius (cylinder) of the current segment being drawn); *level-of-detail* — a normalised scalar value that specifies the output accuracy of all geometric primitives to their ideal forms (Figure 6-2.) This feature is useful for polygon-based rendering systems, which can suffer from performance problems when overloaded with too many polygons in a scene definition.
- A dynamic array of cross-sectional curves (defaulting to the single entry of a unit circle), these curves are used by the generalised cylinder system, described in section 6.6.

[52] Actually, references to materials rather than the materials themselves.

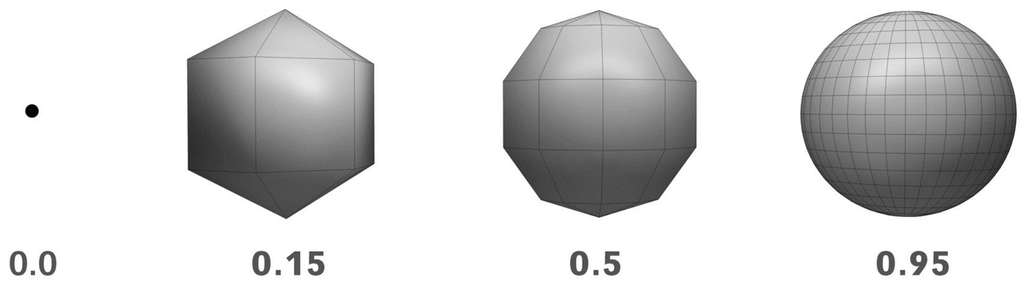


Figure 6-2: A sphere primitive with differing level-of-detail values as shown. The underlying polygonal structure has been superimposed as a vector drawing over a rendered image of each sphere.

- A tropism *elasticity factor* used to model phototropic and geotropic effects when modelling plant structures (see (Prusinkiewicz & Lindenmayer 1990, page 58)).
- Customised state attributes specific to the output device and format. This permits a generalised form of communication with the render to set specific options particular to the rendering system being used.

In the case of parametric L-systems, each symbol has a number of associated parameters. These parameters play an important role in turtle interpretation, controlling the continuous properties of each command (detailed in the following sections).

6.3.1 Instantiating Geometry

The primary purpose of the turtle when building geometric datasets is to instantiate geometric structures at specific locations in three-dimensional space. The turtle position and orientation form a co-ordinate system, $\mathbf{C}_T$, that is always defined relative to a world co-ordinate system, $\mathbf{C}_W$. Placing geometry defined relative to the world co-ordinate system at the turtle co-ordinate system involves a co-ordinate transformation, which can be represented by a homogeneous 4x4 transformation matrix, $\mathbf{M}_{\mathrm{WT}}$, since the transformations are affine. This transformation uses basis vectors to align the orientation of the world coordinate system to that of the turtle coordinate system.

To transform a homogeneous point, $\mathbf{p}_w = \begin{bmatrix} x_W & y_W & z_W & 1 \end{bmatrix}^T$, defined relative to $\mathbf{C}_W$ into its instantiated position at the current turtle co-ordinate system, the following transformation is applied

$$\begin{aligned} \mathbf{p}_T &= \begin{bmatrix} x_T & y_T & z_T & 1 \end{bmatrix}^T \\ &= \mathbf{T}_T \mathbf{M}_{WT} \mathbf{p}_W \\ &= \mathbf{T}_T \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{\mathbf{H}} \\ \vec{\mathbf{L}} \\ \vec{\mathbf{U}} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \end{bmatrix} \mathbf{p}_w \end{aligned} \tag{6.1}$$

This transformation is used when instantiating geometry at the current turtle reference frame.

6.4 Basic Turtle Commands

In order to allow a turtle interpretation, a set of predefined symbols are assigned specific commands. These commands modify the turtle state or signal the construction of geometric structures. In the case of parametric L-systems, each symbol has associated scalar parameters that often control the amount or quality of a particular command. For example, the "+" symbol is interpreted as "turn left", the parameter associated with it specifies by how many degrees to turn. In general, if a parameter is not supplied, a default, globally defined value is used (which may be modified).

A summary of the basic turtle commands is provided in this section. More advanced commands are detailed in sections that follow. The symbol and its associated parameters are shown. Parameters are optional in most cases, so if they are not specified default values are used.

Position and Orientation modifying commands:

Command	*Default*	*Description*
`f(l)`	`l` $= l^*$	Move forward by l units in the direction of $\vec{H}$. The new turtle position $\mathbf{t}' = (t'_x, t'_y, t'_z)$ is calculated $\mathbf{t}' = \mathbf{t} + l\vec{H}$.

[*] Globally defined values that can be changed by the user. Defaults to $l = 1$, and $\delta = 90°$ ($\pi/2$).

Position and Orientation modifying commands:

`+(θ)`	$\theta = \delta^*$	Turn left by angle θ degrees.
`-(θ)`	$\theta = \delta$	Turn right by θ degrees (equivalent to `+(-θ)`)
`&(θ)`	$\theta = \delta$	Pitch down by θ degrees.
`^(θ)`	$\theta = \delta$	Pitch up by θ degrees.
`\(θ)`	$\theta = \delta$	Roll left by θ degrees.
`/(θ)`	$\theta = \delta$	Roll right by θ degrees.
`|(n)`	$n = 1$	Turn around *n* times. Equivalent to `+(180n)`.

Table 6–1: Basic turtle commands affecting turtle position and orientation.

Figure 6-3 illustrates these turtle orientation changes.

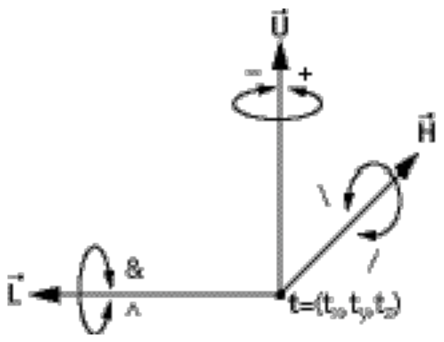


Figure 6-3: Turtle axis and positioning system.

The bracketing commands save and restore turtle state on a stack. Initially they were included as the primary mechanism to model branches directly, since they allow the turtle to return to a parent branch point after a child branch has been constructed. In addition to this function, a number of new capabilities will be described in the following sections.

State save, restore and generate commands:

Command	*Description*
`[`	Save the current turtle state onto a first-in, last-out stack. All components of the state, as specified in Section 6.3, are saved.

State save, restore and generate commands:

]	Restore the current turtle state.
%	If this is the first time this particular symbol has been encountered: interpret current derivation string μ up to the position of this symbol without generating any output. Store turtle reference frame with this symbol. Subsequent readings of the symbol set the turtle reference frame to the value stored with the symbol. This command is primarily used with timed L-systems (Section 8.2).

Table 6–2: Turtle bracket commands.

The "%" symbol is used as a "break" function, breaking the dependency of the turtle reference frame from any previous interpretations that effect position and orientation to the point the symbol was encountered. The primary purpose of this symbol is to enable new geometric sets to move and develop independently of their parents, such as leaves falling from a tree, or flowers emitting seeds. As this is inherently tied to animation, it is detailed in Section 8.3.

Interpreting the string generated from L-system (5.1), at derivation $n_\mu = 3$ gives the image shown in Figure 6-4 below.

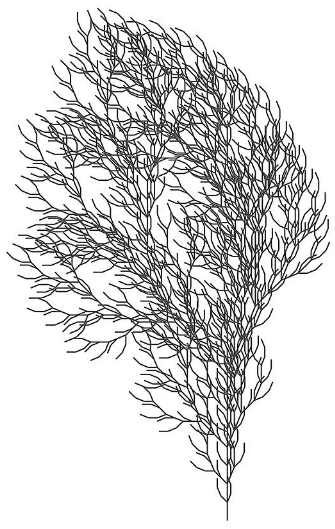


Figure 6-4: Turtle interpretation of the L-system (5.1) from Section 5.3.2. Note that even though this image has a 'tree-like' shape, it is only two-dimensional.

Turtle transform commands provide a mechanism to change geometry — either built-in or externally defined surfaces. These commands change the transform matrix $\mathbf{T}_T$, which is used to transform instanced geometry (see Section 6.3.1). These transformations are always applied relative to the turtle reference frame.

Transformation modification commands:

Command	*Defaults*	*Description*
`S(n)`	`n` = 1	Uniform scale — scale by `n` equally in the x, y, and z axes.
`SX(n)`	`n` = 1	Scale in the direction of the x axis by `n`
`SY(n)`	`n` = 1	Scale in the direction of the y axis by `n`
`SZ(n)`	`n` = 1	Scale in the direction of the z axis by `n`
`SH(a,θ)`	`a` = (0,0,1) $\theta = \frac{\pi}{4}$	Uniform shear by θ about `a`.
`RT`	-	Reset the transformation matrix to the unit matrix, i.e. $\mathbf{T}_T = \mathbf{I}$. This cancels the effect of any transformations.

Table 6–3: Turtle transformation commands.

Turtle instructions can also instance basic geometry that is "built-in" to the system (drawn from internal algorithms). All the primitives described below are subject to the transformations described above allowing an even greater range of geometric shapes to be instanced (for example a scaled sphere will be an ellipse). With the exception of the "`F`" command, none of the basic primitive generation commands change the turtle position or orientation.

Basic Geometric Modelling Commands (Geometric Primitives):

Command	*Defaults*	*Description*
`F(l)`	`l` = l	Draw a line or cylinder beginning at the current turtle position, with length l units in the direction

Basic Geometric Modelling Commands (Geometric Primitives):

		of $\vec{\mathbf{H}}$. The new turtle position $\mathbf{t}' = (t'_x, t'_y, t'_z)$ is calculated $\mathbf{t}' = \mathbf{t} + l\vec{\mathbf{H}}$.
`!(r)`	r = $l/10$	Set the drawing radius to r. A value of 0 means that the F command will draw lines; non-zero values will draw cylinders. The radius command also affects the radius of the default generalised cylinder cross-section (see Section 6.6.3 for details).
`*(n)`	n = 0.5	Set the level of detail for generated geometry to n (where $0 \le n \le 1$, see Figure 6-2). This command affects all instanced geometry generated by the turtle.
`sphere(r)`	r = l	Draw a sphere of radius r, centred at the current turtle position, and aligned with the turtle co-ordinate system.

`box(h,w,d)`	h = l
	w = l
	d = l
		Draw a box with dimensions h, w, d (height, width and depth). The box is centred at the current turtle position and aligned with the turtle co-ordinate system.

`torus(i,o)`	r = $l/10$
	d = $l/2$
		Draw a torus with inner radius i and outer radius o at the current turtle position.

`cone(r,h)`	r = $l/10$
	h = l
		Draw a cone of base radius r and height h. The base is at the current turtle position and the cone is aligned with the turtle co-ordinate system.

`disc(i,o)`	i = 0
	o = l
		Draw a disc with inner radius i and outer radius o at the current turtle position.

Table 6–4: Turtle commands for generating basic geometric primitives.

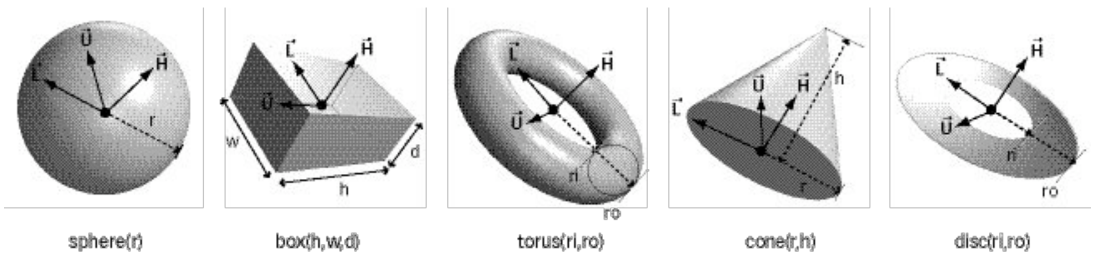


Figure 6-5: Built-in turtle geometric primitive commands and the shapes they produce.

6.5 Surface modelling commands

While the basic built-in primitives described in the previous section give reasonable flexibility in modelling, they are not general enough to cover all possible shapes. Previously described systems allowed the use of pre-defined surfaces (usually parametric bicubic patches) to model petals and other flower elements (Prusinkiewicz & Lindenmayer 1990). Editing of such surfaces is usually achieved with some form of external surface editor with the surface defined relative to a local coordinate reference frame. The surface is then instantiated by the turtle using the "~" command, e.g. "~P" reads a surface called "P".

The approach to external surfaces used here is a natural extension to this system, giving greater flexibility particularly when generating animated sequences (described in Chapters 8 and 9). Surfaces created using some form of external editor can be used by the turtle as long as they are declared in advance. This is achieved by the *surface* directive:

surface surfaceName[(n)]

Where *surfaceName* is the name of the surface to be loaded. This directive automatically adds a symbol of the same name to the alphabet of the current L-system. The optional parameter specifies the number of *vertex sets* to be loaded for that surface (the default value is one). In the case of two or more vertex sets, each set is loaded into a *surface array.* Each surface in the array has the same number of vertices (i.e. the sets have identical order). Surface vertex sets form a multidimensional array and are capable of *weighted interpolation*, permitting "morphing" effects between elements of the surface array.

A surface is instantiated by the turtle simply by its name (no "~" is necessary). This does present the potential for namespace conflicts, so surfaces with names the same as any of the built-in turtle commands are not permitted. An optional parameter controls interpolation of the vertex sets within the surface array.

A surface can be considered an ordered list of vertices, $\mathbf{v}_i$, and associated connectivity and interpolation information. For example, a polygonal mesh is defined by connectivity between vertices using a piecewise linear interpolation. A Bézier surface may be defined by a *natural ordering* of vertices, forming a *Bézier net*, with bicubic interpolation creating the surface (Böhm, Farin & Kahmann 1984). For this discussion, only the vertices are of interest.

Consider a list of surfaces, $S_j, j = 1,\ldots,m$, each with exactly n vertices. A vertex within this list $\mathbf{v}_{ij}, i = 1,\ldots,n$, is the ith vertex of surface S_j. For each surface, we define a scalar weight, $w_j \in [0,1]$, with

$$\sum_{j=1}^{m} w_j = 1 \tag{6.2}$$

When the turtle instantiates a surface, the parameter k controls the blending of surfaces in the array by affecting the weights. Let $S^I(k)$ be the interpolated surface derived from the list of surfaces, S_j. Each vertex, $\mathbf{v}_i^I, i = 1,\ldots,n$ is calculated:

$$\mathbf{v}_i^I = \sum_{j=1}^{m} w_j \mathbf{v}_{ij} \tag{6.3}$$

with the weights calculated using k, where $1 \le k \le m$:

$$w_j = \begin{cases} 1 - \left(k - \lfloor k \rfloor\right) & \text{if } j = \lfloor k \rfloor \\ k - \lfloor k \rfloor & \text{if } j = \lfloor k \rfloor + 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

This simple technique is sufficient for basic transformations and requires the surfaces to be *homeomorphic* (topologically equivalent). It does not explicitly address the *correspondence problem*. A more general transformation technique (Kent, Carson & Parent 1992) could be used to address these limitations.

Surface modelling commands:

Command	*Default*	*Description*
surface`(n)`	`n` = 0	Instance a pre-defined external surface array, *surface*, interpolating to `n`. The surface is transformed to the current turtle position and orientation then any other transformations in the $\mathbf{T}_T$ matrix are applied.
`bP`	-	Begin new polygon — create a new empty polygon and push it onto the polygon stack.
`Pe`	-	End polygon definition — the polygon is drawn and popped off the polygon stack.
`.`	-	Record a vertex for the current polygon at the current turtle position.
`SH(n)`	`n` = 1	Uniform shear by `n`.
`RT`	-	Reset the transformation matrix to the unit matrix, i.e. $\mathbf{T}_\mathrm{T} = \mathbf{I}$. This cancels the effect of any transformations.

Table 6–5: Turtle surface modelling commands.

6.6 Advanced Geometric Modelling Commands

As previously noted, a number of authors have studied the morphology and structure of natural forms in computer graphics. The diversity of such forms represents a significant challenge for computer graphics in terms of developing modelling techniques suitable for describing the immense variety of structures found in nature.

6.6.1 Previous Work

Wainwright suggests that the cylinder has found general application as a structural element in plants and animals (Wainwright 1988). He sees the cylinder as a logical consequence of the *functional morphology* of organisms, seeing the dynamic physiological processes (function) as dependent on form and structure over time. Wainwright distinguishes the cylinder as a natural consequence of evolutionary design based on the physical and mechanical properties of cylindrical structures.

D'Arcy Thompson examined a vide variety of morphological structures in nature, looking at structural similarities (such as logarithmic and natural spirals) that cross species (Thompson 1961). In the turtle interpretations developed by Prusinkiewicz, the cylinder, as a natural extension to the line, is a fundamental geometric construction primitive, with more complex surfaces defined by external modelling tools and instanced as required (Prusinkiewicz 1986b, 1987; Prusinkiewicz, Lindenmayer & Hanan 1988). Hanan discusses a method whereby turtle commands (hence L-system symbols) can be used to describe the control points in a bicubic surface patch, primarily for modelling development of leaves (Hanan 1992, pages 78-84).

Holton developed a botanical tree model where branch segments and joints were modelled by Bézier splines, assuming a circular cross-section (Holton 1994). Control points of the curves were distorted by simulation of a number of competing tropisms and a stochastic perturbation known as the *writhe coefficient*. The use of this coefficient was to produce a more faithful simulation of inter-species variation.

The cylinder, represented by the symbol "F" is often the basic geometric element used in conjunction with the branch-enabling symbols ("[" and "]") to model branches and internodes in three-dimensions with L-systems. This method has its origins in the early days of graphical representations, where most plant imagery was based on line drawings, the line segment of constant thickness being an idealisation of a branch distal segment (e.g. the line drawings based on the results of Hogeweg and Hesper (Hogeweg & Hesper 1974)).

This simple cylindrical method is not sufficient for more complex geometric modelling. Consider the case of modelling a compound segment, such as a tentacle or horn. A relatively simple L-system can be used to describe such a shape, as illustrated in Figure 6-6. The problem in using the cylinder symbol, F, is that complex segments exhibit discontinuities between segments. This problem of describing more complex cylindrical shapes can be solved using *generalised cylinders*, originally developed by Agin for applications in computer vision (Agin 1972). Generalised cylinders have been used extensively by

Bloomenthal to model tree limbs (Bloomenthal 1985) and a variety of natural objects (Bloomenthal 1995).

The *xfrog* system of Lintermann and Deussen makes basic use of cylindrical structures in stem and branch modelling (Lintermann & Deussen 1998, 1999). However, the cylindrical modelling capabilities are not as general as the methods described here. In addition, *xfrog* uses a graphical system for specifying plant construction, where iconic boxes representing preset construction sequences are connected to form a graph. This graph is then interpreted to create the plant model. Direct user specification of L-system grammars is not used; hence the flexibility of integrating generalised cylinder construction commands into the L-system language itself is not possible in the *xfrog* system.

In a recent paper, Prusinkiewicz et. al. describe an interactive plant modelling system that makes use of generalised cylinders and is based on L-systems (Prusinkiewicz et al. 2001). In order to make use of external positional information, the system modifies the traditional parallel application of L-system productions and instead uses a sequential rewriting system based on Chomsky grammars. In this system, generalised cylinders are swept by a sequence of small turtle movements, controlled by three functions which specify the rate of turtle rotation around the principle axes of the turtle reference frame (i.e. about $\vec{\mathbf{H}},\vec{\mathbf{L}},\vec{\mathbf{U}}$). These three functions, which the authors denote $\omega_H(s)$, $\omega_L(s)$ and $\omega_U(s)$ control the bending, twisting and tapering of the cylinder as it is swept out over s. Functions can be specified by a smooth curve $\vec{\mathbf{P}}(s), s \in [0,l]$, usually described via an interactive curve editing program that forms part of the plant modelling system. Curves can be applied to the incremental modification of the turtle reference frame in such a way as to obtain the Frenet reference frame or the parallel transport frame (Bloomenthal 1990).

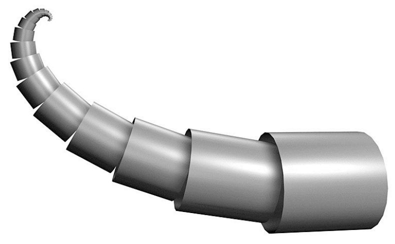
The approach used in (Prusinkiewicz et al. 2001) is similar to that described here[53], however the method developed in this thesis does not rely on external functions to draw curves, rather they are specified by an extended set of turtle commands. Calculation of the Frenet reference frame is achieved using an incremental method. This method does not eliminate the possibility

[53] The method described in this thesis is based on research carried out over the period 1991–1994, prior to the publication of later techniques by other authors.

of inflection points[54] automatically — when inflection points do occur, this is usually a consequence of inconsistencies in the produced string.

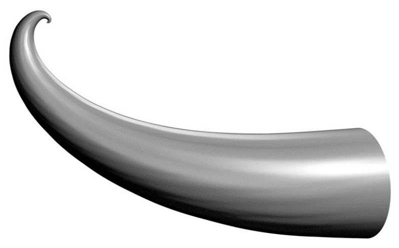
6.6.2 Creating Generalised Cylinders

As discussed, the standard set of turtle commands previously described are incapable of accurately modelling complex shapes such as horns, limbs, leaves, stems and tentacles. Generalised cylinders permit a much richer set of geometric structures to be modelled (Figure 6-6).



```
#define N   10
#define L   10
#define R   5
#define A   10
#define k_l 0.8
#define k_r 0.8
ω : horn(N,L,R,A)
p_1 : horn(n,l,r,a) → seg(n,l,r,a)
p_2 : seg(n,l,r,a) : n > 0 : →
    !(r) F(l) ^(a)
    seg(n − 1, l * k_l, r * k_r, a * 1.1)
```

A



```
#define N   10
#define L   10
#define R   5
#define A   10
#define k_l 0.8
#define k_r 0.8
ω : horn(N,L,R,A)
p_1 : horn(n,l,r,a) → c(1) cseg(n,l,r,a)
p_2 : cseg(n,l,r,a) : n > 0 : →
    !(r) C(l) ^(a)
    cseg(n − 1, l * k_l, r * k_r, a * 1.1)
```

B

Figure 6-6: A simple horn defined (**A**) using cylinders, which leaves noticeable gaps where the radius and angle of the cylinder changes. In **B**, this problem is fixed with the use of generalised cylinders. The parametric L-system generating each model is shown below the image.

The basic principle for creating a generalised cylinder is to define a series of *cross-sections*, possibly of varying shape and size, distributed over some continuous curve, known as the *carrier curve*. The cross-sections are connected to form a continuous surface. This is illustrated in Figure 6-7.

[54] An inflection point is defined as a point on the carrier curve where the tangent intersects that curve. This often corresponds to points of zero curvature (Farin 1990).

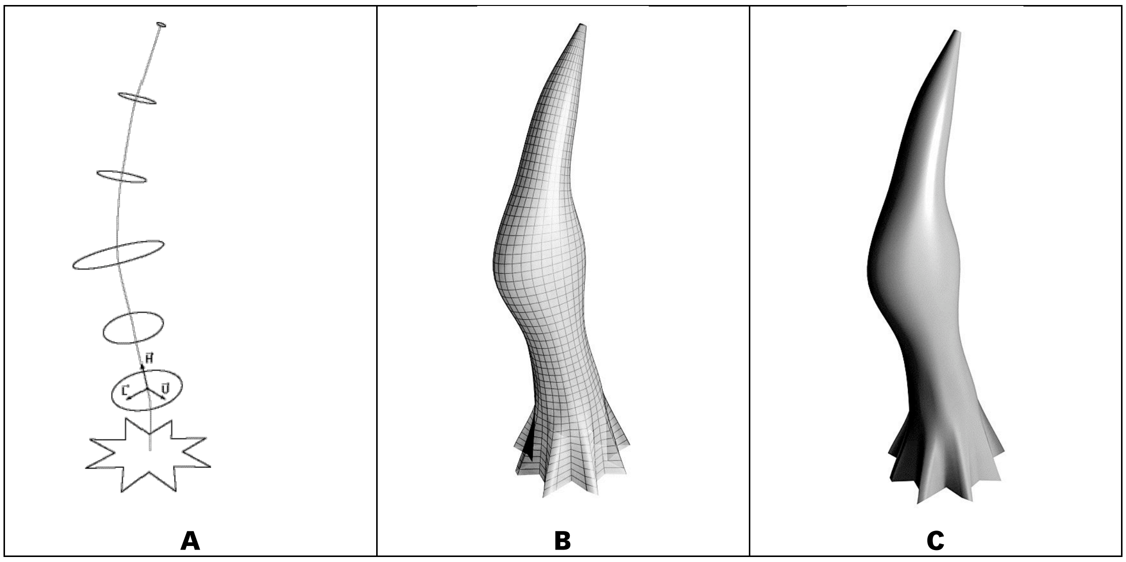


Figure 6-7: Construction of a generalised cylinder. **A** shows the cross-sectional elements and how they are oriented according to a Frenet reference frame, which corresponds to the turtle frame of reference at cross-section instantiations. The path traced out through the cross sections represents the path taken by the turtle as it moves between cross-sections, thus creating a carrier curve. **B** shows the polygonalisation of the cylinder, and **C** a rendered version.

In general, construction of generalised cylinders by this method requires the use of a *reference frame* to avoid improper twisting of the constructed surface. If the surface twists there may be texturing artefacts associated with the twist that are unwanted, or in extreme cases, illegal geometry.

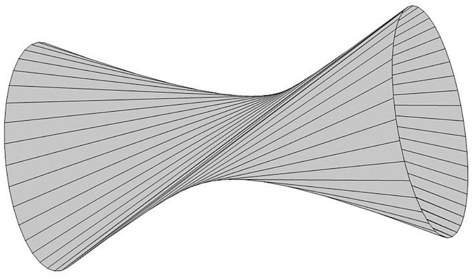


Figure 6-8: Twisted cross-sections.

The Frenet reference frame consists of a position **p** and three orthogonal unit vectors (Faux & Pratt 1979):

$\vec{\mathbf{v}}$, the *velocity* of the curve, corresponding to the tangent

$\vec{\mathbf{n}}$, the *principle normal*, and

$\vec{\mathbf{b}}$, the *binormal*, equal to $\vec{\mathbf{v}} \times \vec{\mathbf{n}}$.

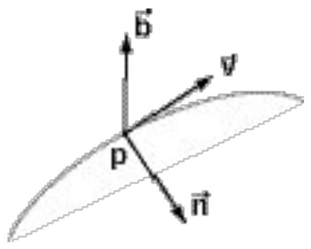


Figure 6-9: The Frenet Reference Frame.

Notice how this reference frame corresponds to the turtle co-ordinate system, $\mathbf{C}_T$, with $\mathbf{p} \equiv \mathbf{t}, \vec{\mathbf{v}} \equiv \vec{\mathbf{H}}, \vec{\mathbf{n}} \equiv \vec{\mathbf{L}}$, and $\vec{\mathbf{b}} \equiv \vec{\mathbf{U}}$. In the normal construction of a generalised cylinder, the principle normal usually points in the direction of curvature of the curve over which the cross-sectional elements are being swept. In the case of defining the cylinder using L-systems, it is up to the grammar to avoid defining cross-sections that cause inflections, hence ensuring a smooth transition from one cross-section to the next. The reference frame is incrementally interpolated (Klok 1986) to ensure a constant rate of change when generating the surface from one cross-section to the next (Figure 6-7). Polygons and texture vertices are generated to try to ensure a constant area for each polygon where possible. The mean area of individual polygons is controlled by the level of detail parameter.

6.6.3 Turtle Commands for Creating Generalised Cylinders

As a natural extension to the simple cylinder, generalised cylinders give a greater modelling flexibility and accuracy when modelling natural forms. Turtle commands to construct generalised cylinders are also a natural extension from the "`F`" command, which creates a simple cylinder. Table 6–6 summarises the turtle commands related to generalised cylinder construction.

Generalised Cylinder Modelling Commands:

Command	*Defaults*	*Description*
`c(s,a)`	`s = 0` `a=FALSE`	Begin generalised cylinder with tangent multiplier `s`. If `a` is `TRUE` then generate a cap (the cross-section is triangulated into a surface).

Generalised Cylinder Modelling Commands:

`C(l,a)`	`l` = l[*] `a=FALSE`	Create generalised cylinder segment of length `l` and move the turtle to position $\mathbf{t}' = \mathbf{t} + l\vec{\mathbf{H}}$ (same as "`F`" command). Equivalent to the sequence: `f(l) Ct(s`$_0$`,a)` Where `s`$_0$ is the value given to the `c` command. If `a` is `TRUE` then generate a cap.

`Ct(s,a)`	`s = 0`
	`a=FALSE`
		Create generalised cylinder segment at current turtle reference frame using tangent multiplier `s`. If `a` is `TRUE` then generate a cap.

`X(n)`	`n = 0`	Select cross-section `n`.
`dX(n)`	`n = 0`	Begin definition of cross-section `n`. (Any existing definition for this index will be replaced). The current turtle reference frame defines the local coordinate system for the cross-section, the $\vec{\mathbf{H}}$ vector defining the cross-section plane normal. This command implicitly saves the turtle state (equivalent to executing a '[' command described in Table 6–2).
`Xd`	-	End definition of current cross-section. The curve is automatically closed (beginning and end points of the cross-section are joined). The turtle state is implicitly restored (equivalent to executing a ']' command described in Table 6–2).
`Xo`	-	End definition of current cross-section. The curve is left open. This is useful for modelling leaf shapes. The turtle state is implicitly restored.
`.`	-	Record vertex for current cross-section at current turtle position.

Table 6–6: Turtle commands for the creation of generalised cylinders.

[*] Globally defined value that defaults to 1, but can be changed by the user.

Internally, the turtle maintains an array of cross-sections. Commands to construct cross-sections will be detailed shortly. A default circular cross-section is created upon initialisation. Generalised cylinders begin with the `c` command, which starts a new cylinder using the currently selected cross-section. Subsequent calls to either the `C` or `Ct` commands create segments. Calls to save and restore turtle state also store the last selected cross-section to permit branching (Figure 6-13).

Interpolation between segments can follow a linear or Bézier path, with the tangent values at each point on the spline determined by the tangent vector ($\vec{\mathbf{H}}$) and the tangent multiplier provided as an optional parameter to the `c`, and `Ct` commands. A value of 0 for the multiplier produces a linear interpolation, 1 gives a natural spline, higher values attenuate the shape of the curve (Figure 6-10).

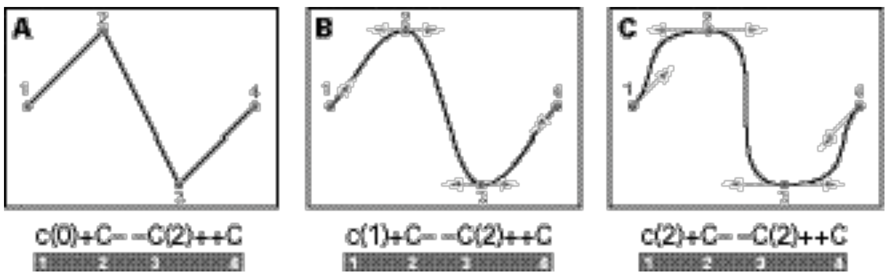
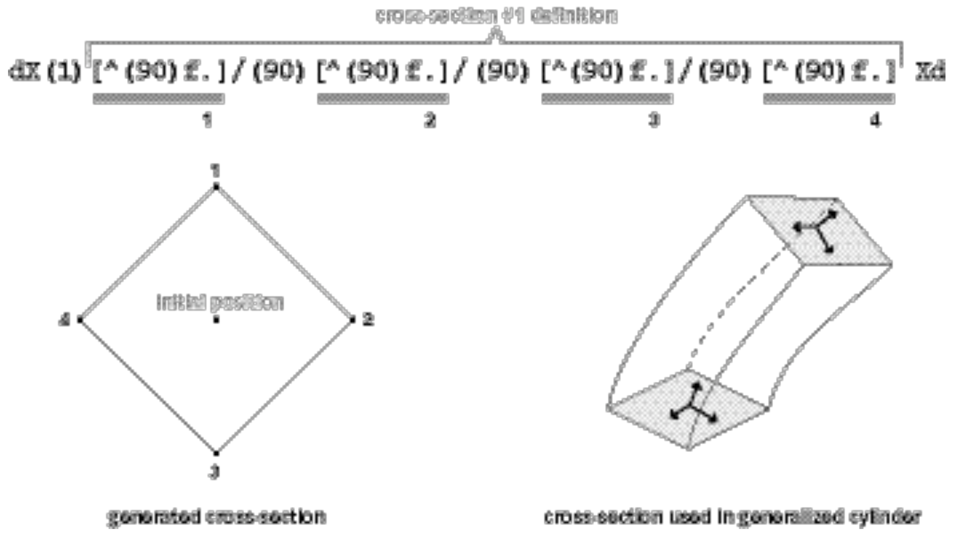


Figure 6-10: The effect of the tangent multiplier, passed as the parameter to the `c` command. A value of 0 produces a linear interpolation (**A**), higher values change the shape of the curve (**B** and **C**).

Cross-sections can be defined in a similar way to that of polygons (Section 6.5). For example, the sequence shown in Figure 6-11 instructs the turtle to create a cross-section to be stored at index 1 in the cross-section array maintained by the turtle. A square is defined in this case, however no actual square is drawn until the cross-section is made active and generalised cylinder drawing commands are issued. The current turtle reference frame defines the local coordinate reference frame for the cross-section. This frame is established at the time the `dX` command is processed, with the $\vec{\mathbf{H}}$ vector defining the normal of the plane within which the cross-section typically lies (it is up to the designer of the L-system productions to ensure that cross-section definitions are planar — nothing internally in the system enforces this). The `dX` and `Xd` or `Xo` command pairs have the side effect of pushing and popping the turtle state, ensuring that the turtle state is unchanged by any cross-section definition. The `Xo` command defines open cross-sections, useful for the modelling of leaves and other shapes without an interior.

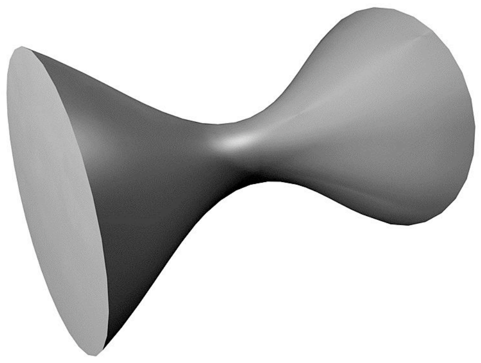


```
dX (1) [^ (90) f. ] / (90) [^ (90) f. ] / (90) [^ (90) f. ] / (90) [^ (90) f. ] Xd
```

cross-section #1 definition
1
2
3
4
initial position
1
2
3
4
generated cross-section
cross-section used in generalized cylinder

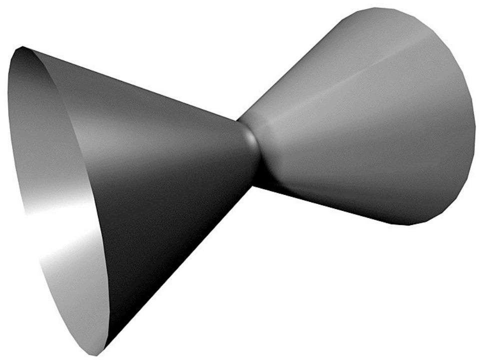
Figure 6-11: Turtle commands to generate a simple cross-section.

Once defined, cross-sections are made active using the `X` command. Only one cross-section may be active at any given time. Following activation of a cross-section, subsequent calls to create generalised cylinder segments will use the active cross-section. The cross-section is placed by aligning its local reference frame (established at definition) with the current turtle reference frame (at the time a `c`, `C` or `Ct` command is processed). The active cross-section may be changed during construction, some examples of this are shown in Figure 6-12. In each case, the necessary derivation length, *n* is shown.



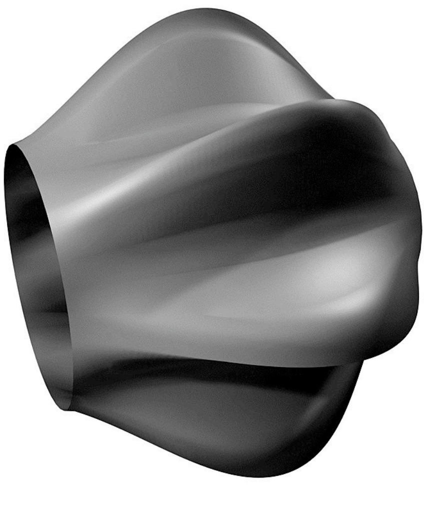
```
$\omega : gs$
$p_1 : gs \to X(0)$  // use default circle
                  // as x-section
    $S(1)\ c(1, TRUE)$ // cap, tangent 1
    $S(0.2)\ C(0, FALSE)$ // scale default circle
    $S(1)\ C(0, TRUE)$

$n = 1$
```



```
$\omega : gs$
$p_1 : gs \to X(0)$  // use default circle
                  // as x-section
    $S(1)\ c(0, FALSE)$ // no cap, linear interp
    $S(0.2)\ C(0, FALSE)$ // scale default circle
    $S(1)\ C(0, FALSE)$

$n = 1$
```



```
    #define R1   1  // inner radius
    #define R2   2  // outer radius
    #define NP   6  // number of points
$\omega : gs$
$p_1 : gs \to dstar\ dcyl$
$p_2 : dstar \to dX(1)\ star(180/NP, 0)\ Xd$
$p_3 : star(\theta, \phi) : \phi < 360 : \to$
    $[+(90)\ f(R1)\ .\ ]\ /(\theta)$
    $[+(90)\ f(R2)\ .\ ]\ /(\theta)$
    $star(\theta, \phi + 2\theta)$
$p_4 : dcyl \to X(0)$  // select circle
    $c(1)$  // begin cylinder
    $X(1)$ // select star
    $C(1)$  // draw segment
    $X(0)$  // select circle
    $C(1)$  // draw segment

$n = 9$
```

Figure 6-12: Example generalised cylinders and their generating L-systems.

The use of generalised cylinders provides a more flexible approach to modelling than that permitted with simple cylinders. The use of Bézier interpolation also permits C1 continuity at branching points — a useful feature for organic modelling (De Leon 1991).

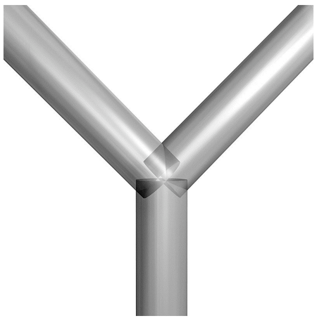
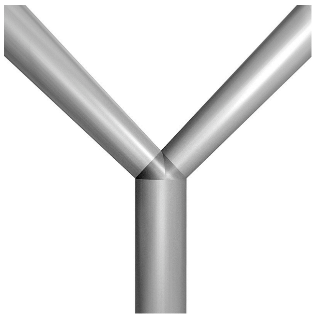
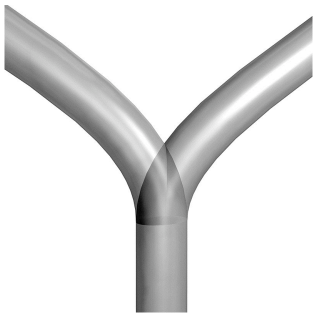
 F[+F][–F]	 c(0)[+C][–C]	 c(1)[+C][–C]
A	**B**	**C**

Figure 6-13: Three different methods of forming branching segments. In each image, the geometry is rendered with a slight transparency to better illustrate the underlying structure. The string below each image is the sequence of turtle commands used to create the segments shown. In the case of **A**, branching segments are modelled using individual cylinders, resulting in discontinuity at the branching points. For **B**, a generalised cylinder with linear interpolation between cross-sections is used, resulting in a join at the branching point. For **C**, a generalised cylinder with C1 continuity gives a smooth transition at branching points.

6.6.4 Cylinder Construction and Calculation of Intermediate Geometry

When using the generalised cylinder turtle commands, the sub-system processing those commands must perform a number of tasks in order to output the geometric representation of the cylinder. We will consider the task of constructing a segment of the cylinder between two instantiated cross-sections (which may be different). This may be decomposed into a series of sub-tasks:

- generation of the carrier curve between the two cross-sections;
- calculation of the reference frame over the carrier curve;
- generation of intermediate cross-sections which smoothly interpolate between the profiles of the two instantiated cross-sections;

- linking the instantiated and intermediate cross-sections to form the complete geometry.

This process is illustrated in Figure 6-14.

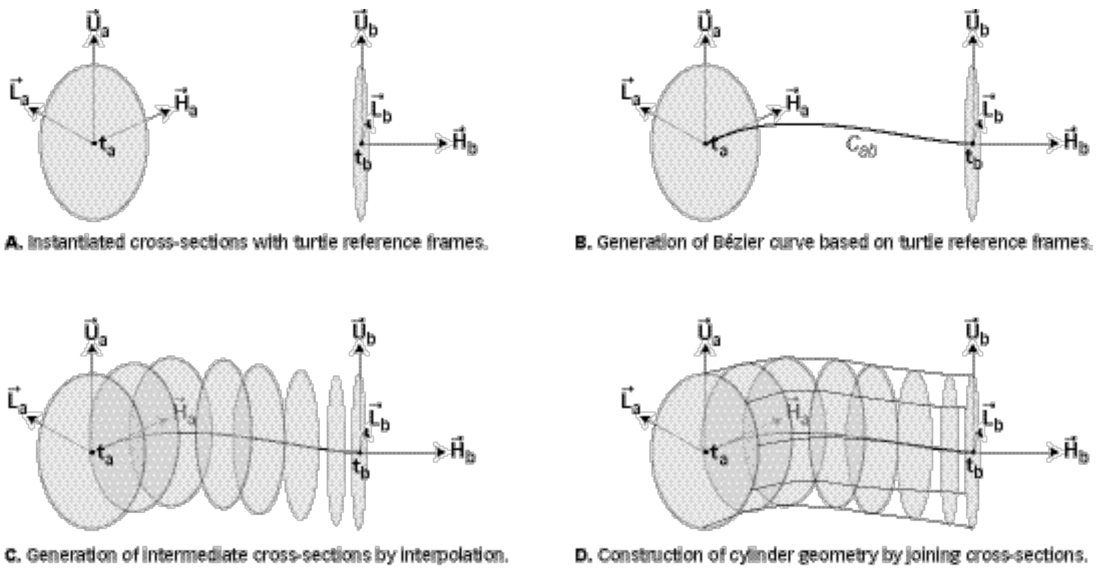

$\vec{U}_a$
$\vec{L}_a$
$\vec{H}_a$
t_a
$\vec{U}_b$
$\vec{L}_b$
$\vec{H}_b$
t_b
C_{ab}
A. Instantiated cross-sections with turtle reference frames.
B. Generation of Bézier curve based on turtle reference frames.
C. Generation of intermediate cross-sections by interpolation.
D. Construction of cylinder geometry by joining cross-sections.

Figure 6-14: Stages of construction of a generalised cylinder segment.

Each of these stages will now be explained in more detail. As shown in the figure, by interpreting the turtle commands there are two cross-sections with their own coordinate reference frames, subscripted **a** and **b**. From these two reference frames, a polynomial curve can be constructed using Hermite interpolation on the turtle positions and heading vectors, i.e. $\left(\mathbf{t}_a, s_a\vec{\mathbf{H}}_a, \mathbf{t}_b, s_b\vec{\mathbf{H}}_b\right)$, where s is the scaling parameter used when the cross-section was instantiated by the `c`, `C` or `Ct` commands (refer to Table 6–6). This carrier curve is defined $C_{ab}(t), t \in [0, l]$, where l is the length of the curve.

Hermite interpolation is used for convenience as the turtle reference frame readily provides the necessary interpolation information. Cubic Hermite interpolation is not, in general, invariant under affine domain transformations (Farin 1990, section 6.5), however as the tangents are calculated in world coordinate space, they are not subject to any additional transformations before the curve is generated, so this variance is not a practical problem. It is also possible to convert to cubic Bézier form, which is invariant if required (Farin 1990).

New cross-sections need to be created over the curve C_{ab}, so it is subdivided according to arc length and rate of curvature (these parameters form part of the turtle state properties (Section 6.3)). Intermediate reference frames are calculated this way.

The cross-sections defined at the endpoints may be different. If this is the case, intermediate cross-sections must "morph" between the two shapes. This is achieved using the methods detailed in Section 6.5 to interpolate surfaces.

Finally, each cross-section is connected together to form the complete cylinder by creating triangles between each cross-section.

C^1 continuity between sets of generalised cylinders is achieved by matching tangents between sets of cross-sections. This ensures a continuous curve over whatever length is required.

6.6.4.1 Convolution Surfaces

An alternate approach to achieving continuity at branching points is to use the generating L-system as a *skeleton* and *convolve* this skeleton to form a three-dimensional surface. In this case, the skeleton consists of connected and branching line segments, such as those created by the turtle `F` command with a radius of 0. As the name implies, convolution surfaces *convolve* a geometric element with a convolution kernel (Bloomenthal & Shoemake 1991). They are a natural extension to the "blobby" primitives described by Blinn (Blinn 1982) and Nishimura (Nishimura et al. 1985).

It is possible to apply a convolution kernel to geometric primitives such as points, lines and polygons and generate an analytic solution suitable for direct rendering (McCormack & Sherstyuk 1998). Figure 6-15 illustrates a simple example using the same branching structure illustrated in Figure 6-13. Note the geometry at the intersection of the branching point and the endpoints of the lines. In this case, generation is a two-step process — first the L-system generates a skeletal model, which is then processed by the convolution surface renderer to create a three-dimensional surface representation. This surface is rendered using the ray-tracing algorithm described in (McCormack & Sherstyuk 1997).

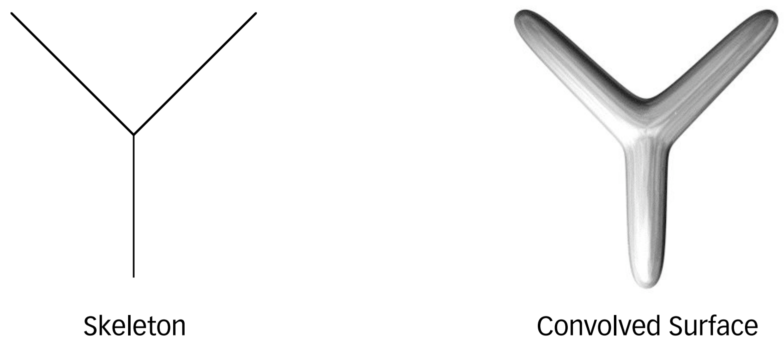


Figure 6-15: L-system generated skeleton and convolved surface generated by convolving the skeleton with an exponential kernel.

6.7 Texture Mapping

In addition to the generation of geometric information, texture information is also needed for image synthesis. Texture mapping is a common technique in computer graphics that transforms information from one coordinate space to another. It is used to provide more surface detail, using standard shading models such as Phong shading. Texture mapping can often give the impression of increased complexity in models without the need for extra geometry.

There are two basic types of texture mapping: two-dimensional and three-dimensional. We will consider two-dimensional first.

6.7.1 Generating 2D textures

Two-dimensional (2D) textures map from a two-dimensional space to a three-dimensional one. This is usually in the form of a raster image, whose channels can be used to control various surface shading parameters such as ambient, diffuse, and specular reflections, transparency, and surface normal perturbation (known as bump mapping). Not only shading channels can be affected by the texture map — in the case of *displacement mapping* the texture displaces geometry, moving the surface from its default position according to the value in the texture channel.

Two-dimensional texture maps are typically defined $T(s,t)$, where s and t form an orthogonal Cartesian local co-ordinate system. As shown in Figure 6-16, the input image is usually normalised so that it spans a unit square (Williams 1983). In the case of parametric surfaces, there is a natural mapping between texture and geometric surface. In the case of polygonal models, *texture vertices* must be generated with each corresponding geometric vertex. For each geometric vertex, $\mathbf{v}_i = (x_i, y_i, z_i) \in \Re^3$, an associated texture vertex, $\mathbf{u}_i = (s_i, t_i) \in \Re^2$ is created.

For the generalised cylinder, the generation of texture vertices proceeds as follows. First, each cross-sectional element is parameterised in s in the interval [0,1]. Texture vertices for each are assigned in direct proportion to the geometric distance between intermediate points, to minimise distortions in the mapped texture. Parameterisation in t follows a similar methodology, starting at 0 when the turtle generates the first element via the "C" command. To maintain a uniform distribution of texture over the cylinder's surface, the values of t at each subsequent cross-section are calculated based on the geometric circumference of that cross-sectional element. For a circle, the circumference is $2\pi R$ where R is the radius of the circle. The value of t is calculated to preserve a 1:1 correspondence between s-distance and t-distance, so for a straight cylindrical element with a constant radial value, R and length H, the value of t at the end of the segment is $\frac{H}{2\pi R}$ (see Figure 6-16).

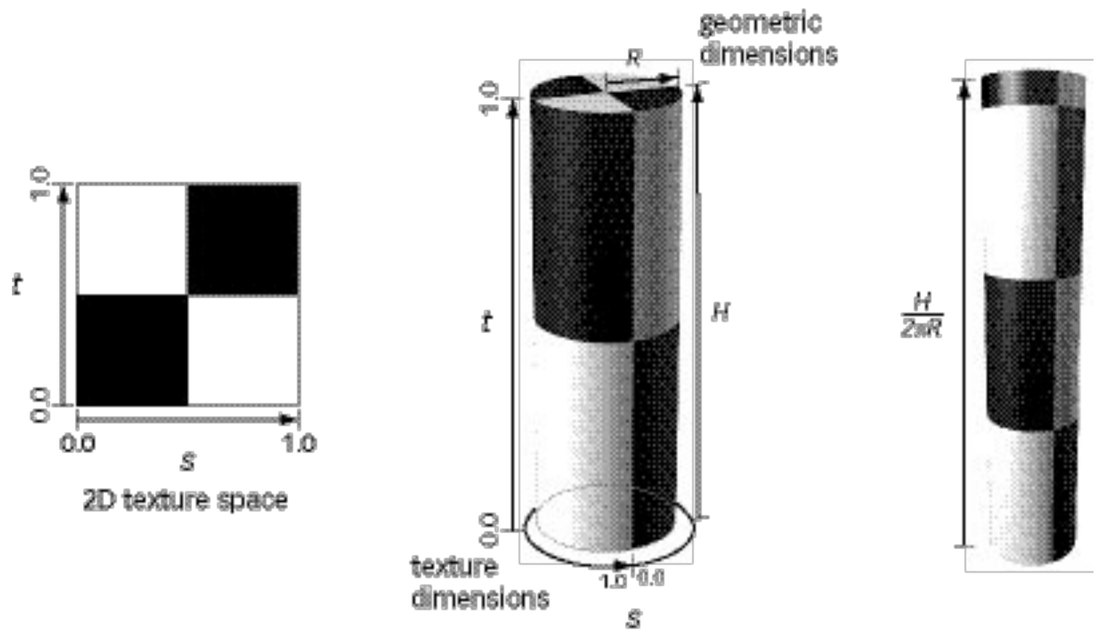
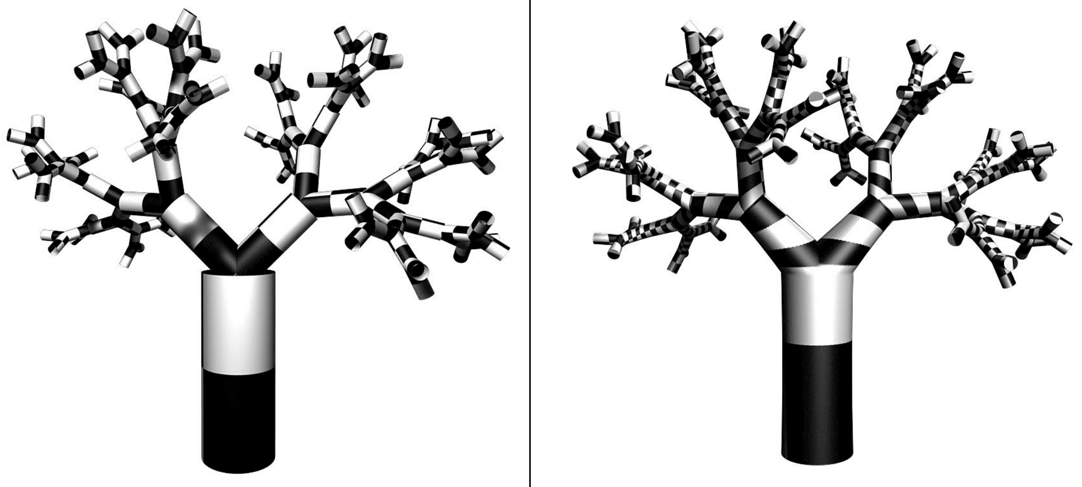


Figure 6-16: Two dimensional texture mapping. An image texture (a chequerboard pattern in this case), defined in 2D texture space is mapped onto the cylindrical element as shown. In the case of generalised cylinder generation, mapping is applied to preserve a uniform texture area over the geometric surface in which the texture is defined.

In more complex cases, textures are calculated by bi-linear or bi-cubic interpolation, based on the principle of maintaining minimal distortions in the applied texture. Figure 6-17 shows this technique on two "tree-like" structures, with a chequerboard pattern applied.



$$
\begin{aligned}
&\#\text{define N} \quad 10\\
&\#\text{define R} \quad 10\\
&\#\text{define L} \quad 50\\
&\delta = \pi/4\\
&\omega : \mathit{tree}\\
&p_1 : \mathit{tree} \rightarrow \mathit{seg}(1)\\
&p_2 : \mathit{seg}(n) \ : \ \mathrm{n} < \mathrm{N} \ : \ \rightarrow\\
&\quad !(R/n) \, / \, F(L/n) \, [+\, \mathit{seg}(n+1)]\\
&\quad [-\, \mathit{seg}(n+1)]\\
&p_3 : \mathit{seg}(n) : n \geq N \rightarrow F(L/n)
\end{aligned}
$$

A

$$
\begin{aligned}
&\#\text{define N} \quad 10\\
&\#\text{define R} \quad 10\\
&\#\text{define L} \quad 50\\
&\delta = \pi/4\\
&\omega : \mathit{tree}\\
&p_1 : \mathit{tree} \rightarrow !(R)\, c\; \mathit{seg}(1)\\
&p_2 : \mathit{seg}(n) \ : \ \mathrm{n} < \mathrm{N} \ : \ \rightarrow\\
&\quad !(R/n) \, / \, C(L/n) \, [+\, \mathit{seg}(n+1)]\\
&\quad [-\, \mathit{seg}(n+1)]\\
&p_3 : \mathit{seg}(n) : n \geq N \rightarrow C(L/n)
\end{aligned}
$$

B

Figure 6-17: Simple tree like structure with texture mapping (the chequerboard pattern as shown in the previous figure). **A** constructed with isolated simple cylinders, and **B** with generalised cylinders.

6.8 Special Functions for Parametric L-systems

Parametric L-systems, defined in Section 5.3.5, associate a set of scalar parameters with symbols in the L-system alphabet. A standard set of arithmetic functions (e.g. +, –, /, *) permit basic mathematical operations to be performed on parameters. In addition, a number of standard mathematical functions are provided (in each case it is assumed x, y and c are real numbers):

- ***Trigonometric functions:*** `sin(x)`, `cos(x)`, `tan(x)`; inverse trigonometric functions: `arcsin(x)`, `arccos(x)`, `arctan(x)`.
- ***Power and logarithmic functions:*** `pow(x,y)` returning x^y, `log(x)`, `ln(x)` (base 10 and natural log functions).
- ***Logical functions and other functions:*** `if(c,x,y)` returns x if $c \neq 0$, otherwise y; `sign(x)` returns 1 if $x > 0$, 0 if $x = 0$, –1 if $x < 0$; `floor(x)` rounds x to the largest integral value not greater than x; `ceil(x)` returns the smallest integral value greater than x; `abs(x)` returns the absolute value of x.
- ***Other mathematical functions*** such as Bessel and hyperbolic trigonometric functions — these will be described in relation to specific applications.

The remainder of this section briefly details other functions developed for the system described in this thesis. These new functions can be divided into three groups:

- ***Interpolative functions*** — perform various types of interpolation.
- ***Pseudorandom and noise functions*** — generate sequences of noises and pseudorandom numbers described by a variety of statistical distributions.
- ***Environmental functions*** — functions that query the environment or provide input data from external sources, such as sound.

These functions are briefly described here. Examples of their use can be found in the next chapter. Many of these functions have application for generating animation, the subject of Chapter 8.

6.8.1 Interpolative Functions

Interpolative functions generate intermediate values between two supplied coordinates. The functions listed here are inspired by those provided in the *RenderMan* shading language (Upstil 1990) and those described by Darwin Peachey in (Peachey 1994) for the purposes of procedural texture synthesis. As will be demonstrated in latter chapters, they can be effectively applied to animation synthesis as well.

Interpolative functions

Function	*Description*
`step(a,x)`	Returns 0 if `x` < `a`; 1 if `x` ≥ `a`.
`pulse(min,max,x)`	Returns 1 if `min` ≤ `x` ≤ `max`; 0 if `x` < `min` or `x` > `max`.
`clamp(x,min,max)`	Returns `x` if `min` ≤ `x` ≤ `max`; `min` if `x` < `min`; `max` if `x` > `max`.
`linear(min,max,x)`	Linear interpolation: returns `min` + (`x` `*` (`max` – `min`)), where normally 0 ≤ `x` ≤ 1.
`linearstep(min,max,x)`	0 if `x` < `min`; 1 if `x` ≥ `max`, otherwise a linear interpolation between 0 and 1.
`smooth(min,max,x)`	Hermite interpolation between `min` and `max` with velocity at the endpoints 0 (often known as an 'ease-in, ease-out' curve).
`smoothstep(min,max,x)`	0 if `x` < `min`; 1 if `x` ≥ `max`, otherwise a Hermite interpolation between 0 and 1.
`spline(v,` k_0, k_1, k_2, `...,` k_n`)`	Return a point on the curve at `v` fitted to the supplied knot values using a one-dimensional Catmull-Rom spline. The spline function takes a series of values (either scalar or vector) representing the equi-distributed knots of the spline over the [0,1] interval. At least 4 knots are required (Ebert et al. 2003, p. 34).

Table 6–7: Interpolative functions.

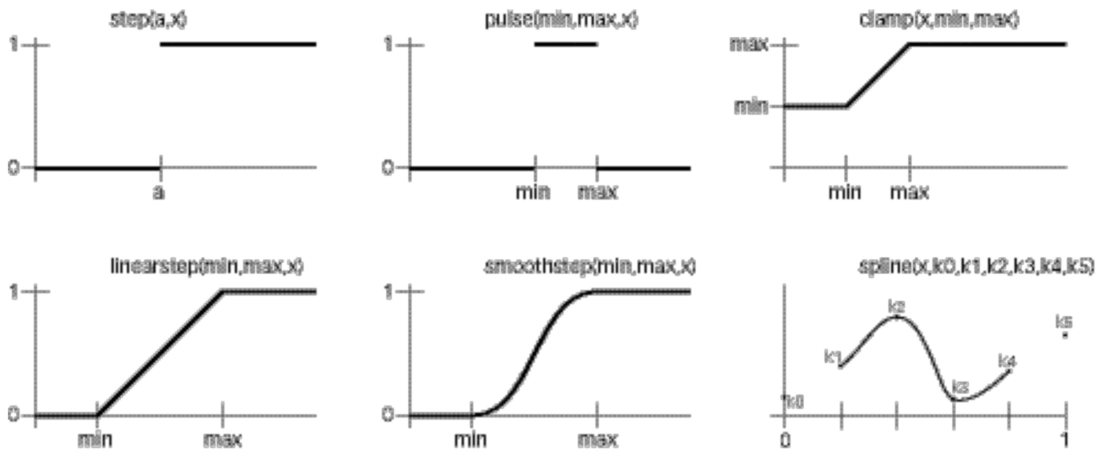


Figure 6-18: Plots of some of the functions defined in Table 6–7.

6.8.2 Pseudorandom and Noise Basis Functions

Random functions provide for a variety of statistical distributions and noise like basis functions. Functions come in two "flavours" — ***keyed*** and ***non-keyed***. Keyed functions take a value as a key, which is used to calculate that function's resultant value. A function called with the same key will always return the same value. This is necessary for animation coherence, where the same "random" number may be required for coherent properties in animated sequences. Functions can be keyed by numeric value (floating point number or vector) or by L-system symbol. Keyed functions will be indicated by a subscript representing the parameter or symbol to which the function is keyed. E.g.,

$$\texttt{random}(\alpha_1, \alpha_2) \tag{6.5}$$

is the non-keyed version of the uniformly distributed random number function, which returns a pseudorandom number uniformly distributed in the range $[\alpha_1, \alpha_2]$, and

$$\texttt{random}_S(\alpha_1, \alpha_2) \tag{6.6}$$

is the keyed version of that function, keyed to the symbol S. Successive calls to $\texttt{random}(\alpha_1, \alpha_2)$ will return a possibly different number each time, whereas successive calls to $\texttt{random}_S(\alpha_1, \alpha_2)$ by the same instance of a symbol S in the derivation string μ_i will return the *same* value at each call. In any given derivation word there may be multiple instances of the same symbol, and the keying is to a specific symbol, not all instances of that symbol across the derivation word (i.e. the keying takes place when a module in the successor of a production is bound to an actual module). The function

$$\texttt{random}_{\mathbf{p}}\left(\alpha_1,\alpha_2\right) \tag{6.7}$$

is keyed to a given three-dimensional point, $\mathbf{p}=\left(p_x,p_y,p_z\right)$. When called for two points with identical locations in space, the pseudorandom number returned will be the same. Points with different values will return (potentially) different pseudorandom numbers.

For clarity, only the non-keyed versions are detailed in Table 6–8, the keyed versions being identical in function with the exceptions and interpretations noted above. The use and significance of keyed functions is further discussed in Section 7.3.

Pseudorandom and Noise functions

Function	*Description*
`random(min,max)`	Returns a uniformly distributed pseudorandom double precision floating point number between `min` and `max`.
`irandom(min,max)`	Returns a uniformly distributed pseudorandom integer between `min` and `max`.
`gaussian(`μ,σ`)`	Returns a normally (Gaussian) distributed pseudorandom number with mean μ, and standard deviation σ. The normal distribution density function is defined as $g(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]}$. This function is calculated using the Box-Muller method (see (Press et al. 1992, Chapter 7)).
`poisson(`μ`)`	Return a pseudorandom number with Poisson distribution with mean μ. The Poisson distribution function is $p(x)=\frac{\mu^x}{x!}e^{-\mu}$. Generated using a variation on the rejection method (Devroye 1986).

Pseudorandom and Noise functions

`noise3D(x,y,z)`
`noise2D(x,y)`
`noise1D(x)`
	A "Perlin" basis noise function (Perlin 1985a). The noise function used is an approximation to low-pass filtered white noise, resulting in a smooth, band-limited and continuous pseudorandom function with a maximum frequency of approximately 1 (see (Peachey 1994) for implementation details). The `noise` function is ***stationary*** (translational invariance) and ***isotropic*** (rotational invariance). One- and two-dimensional versions of the `noise` function and its related functions are also provided. All noise functions return a scalar value in the range [-1,1].

`dnoise3D(x,y,z)`	Instantaneous vector differential of the `noise3D` function at position ***(x,y,z)***.

`turbulence(x,y,z, octaves)`	Turbulence function calculated by summing noise values at different frequencies. E.g.:
	`turb(x,y,z,octaves) {`
	`   double f, value = 0.0;`
	`   for (f = 1; f<=octaves; f*=2)`
	`     value +=   abs(noise3D(x*f,y*f,z*f))/f;`
	`   return value;`
	`}`

`fractalSum(x,y,z, octaves)`	Identical to the `turbulence` function above with the exception of summing the explicit (rather than absolute) values of the `noise` function.

Table 6–8: Pseudorandom and Noise functions.

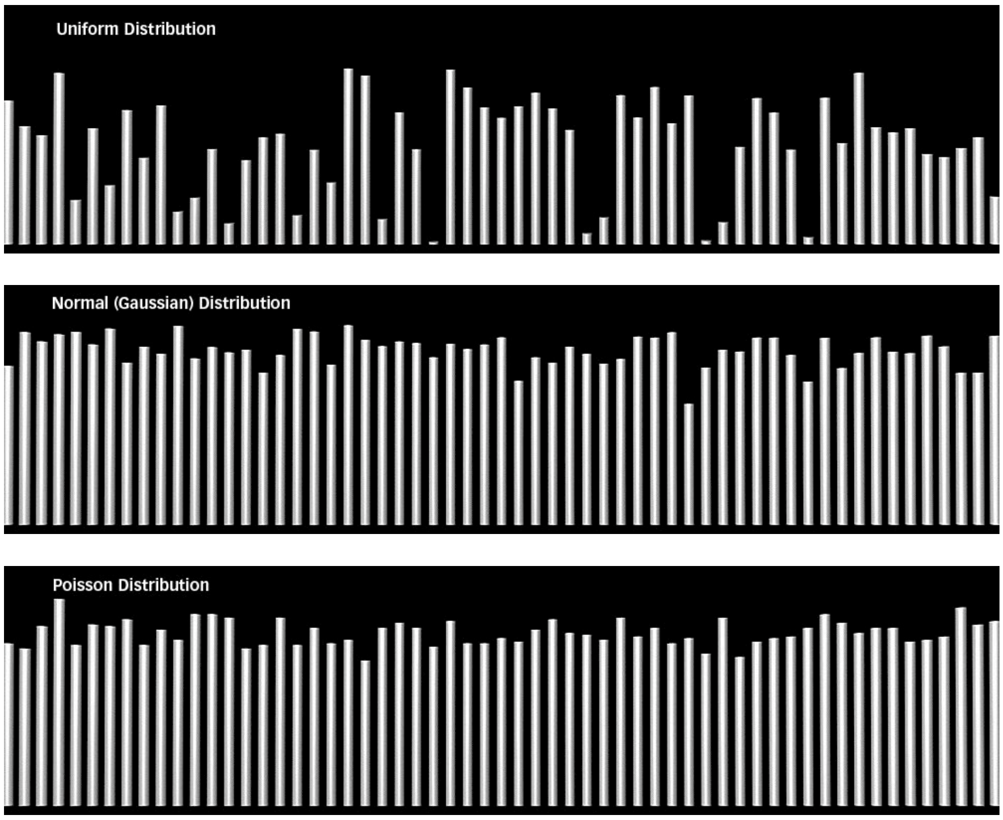


Figure 6-19: Cylinders with randomly varying heights. Each cylinder is an instance of the turtle ‘`F`’ symbol with parameter value supplied by the `random(0,1)`, `gaussian(1,0.1)` and `poisson(1)` functions described in Table 6–8.

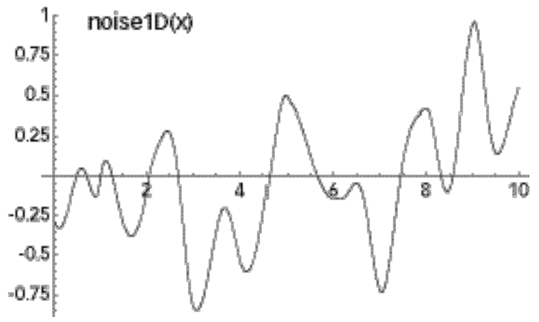


Figure 6-20: Graph of the noise function in one dimension.

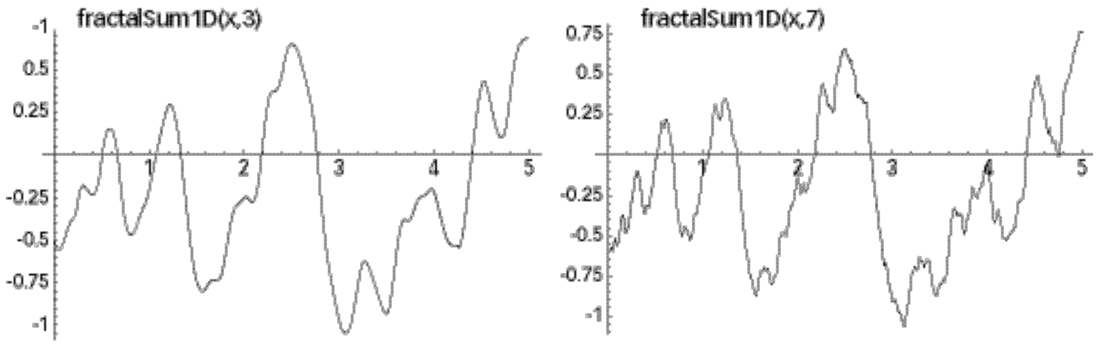


Figure 6-21: Graph of the `fractalSum` function in one dimension with 3 and 7 octaves.

6.8.3 Environmental Functions

Environmental functions allow L-system symbols to query physical or environmental properties that are external to the L-system itself. This allows, for example: detection of intersection with external geometry; movement and growth driven by external data, such as sound; specification of development functions (refer to Section 8.2.1) from external data sources, such as statistical samples. The reporting of turtle position is similar to that described in (Mech & Prusinkiewicz 1996).

Environmental functions

Function	*Description*
`tPosition()`	Returns the current geometric position vector of the turtle, **t**, at the time the function was called (refer Section 6.3).
`tHeading()` `tLeft()` `tUp()`	Returns the turtle $\vec{\mathbf{H}}, \vec{\mathbf{L}}, \vec{\mathbf{U}}$ vectors respectively (defined in Section 6.3).
`data(filename,pos)`	Returns the numeric value of the ASCII data stored in the file `filename` at line number `pos`. Typically such files would have lines of floating point numbers representing some data vector created by an external source (for example motion-capture data or an external curve editor).

Environmental functions

`sound(filename,t)`	Returns the normalised amplitude (envelope) of the audio sample file, `filename` at time `t` seconds. The file is assumed to contain digitised audio samples in the AIFF format. This function can be used to animate parameters to soundtracks, facilitating for example, lip-synced movement to a voice track.
`ffta(filename,t,f)`	Returns the approximate relative amplitude of sound in an audio file around `f` hertz about the time `t` seconds. In basic terms, this involves taking the Fourier transform of the audio file and calculating the relative energy at the supplied frequency at the requested time. Phase information is not returned. Relative energy for each frequency is averaged over a small time quantum, usually a few milliseconds. See (Roads 1996, Appendix) for details.

Table 6–9: Environmental Functions.

6.8.4 Summary

The use of functions described in this section describes versatile and flexible methods to permit L-system to be used for a diverse range of modelling problems. The application of these functions to modelling problems in computer graphics will be detailed in the next chapter.

7 Applications

The power of the golden section to create harmony arises from its unique capacity to unite different parts of the whole so that each preserves its own identity, and yet blends into the greater pattern of a single whole.

— György Doczi, The Power of Limits (Doczi 1981)

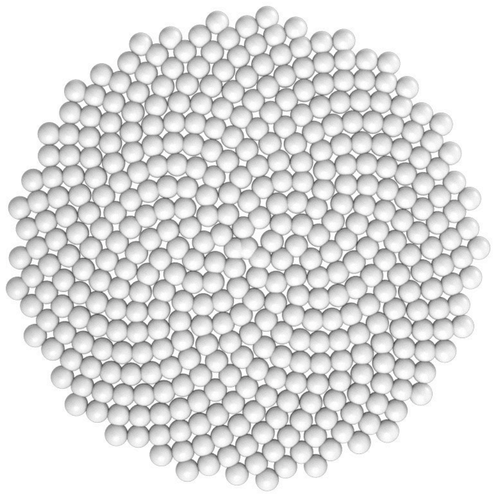
Having described the key features of a generative modelling system based on L-systems in the previous chapter, this chapter gives some practical examples of how the system can be used to model organic form. A generalised method for generating phyllotaxis on surfaces of revolution extends the work done by previous authors in using L-systems to model phyllotaxis. This section describes an *area-based phyllotactic model*, developed as part of my research. This model permits placement of small elements in phyllotactic patterns over surfaces of revolution. The model itself is described in Section 7.2.

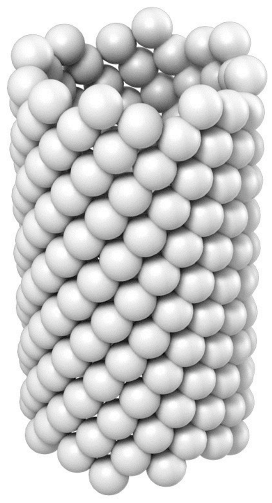
This chapter also includes some examples of natural forms modelled using the generalised cylinder techniques, illustrating the flexibility permitted by this extension and the broad gamut of modelling possibilities for L-systems. Finally, the use of stochastic functions is illustrated in modelling organ and inter-species variation.

7.1 Phyllotaxis

Phyllotaxis is the regular arrangement of plant lateral organs according to an optimal packing scheme, based on the golden ratio. It is an interesting "self-organizing" property of plant morphogenesis, demonstrated experimentally (Douady & Couder 1992) and computationally (Douady & Couder 1996). Examples include leaves on a stem, cone scales on a cone axis, and florets or achenes in a composite flower head. Phyllotaxis has been extensively studied in the literature from a variety of perspectives that include art, architecture, mathematics, biology and computer graphics (a recent overview can be found in (Jean & Barabe 1998), and Ball gives an overview from a developmental morphogenesis perspective (Ball 2001, Chapter 4).

Helmut Vogel was one of the first to develop a planar mathematical model of phyllotaxis based on florets of equal area (Vogel 1979). The "numbers of spirals that can be traced through a phyllotactic pattern are predominantly the integers of the Fibonacci sequence" (Erickson 1983, page 54). Phyllotactic patterns arrange the lateral organs in ranks or *parastichies*. In geometric modelling of phyllotaxis using L-systems, lateral organs are typically distributed over cylindrical or planar surfaces (Prusinkiewicz & Lindenmayer 1990, Chapter 4). Figure 7-1 shows examples of these placements and the L-systems that generated them.





Planar Phyllotaxis:

```
    #define a   137.5
    #define N   400
$\omega$ : A(0)
$p_1$ : A($n$) : $n < 400 \rightarrow$ +(137.5)
    [ f($\sqrt{n}$) sphere ] A($n+1$)
```

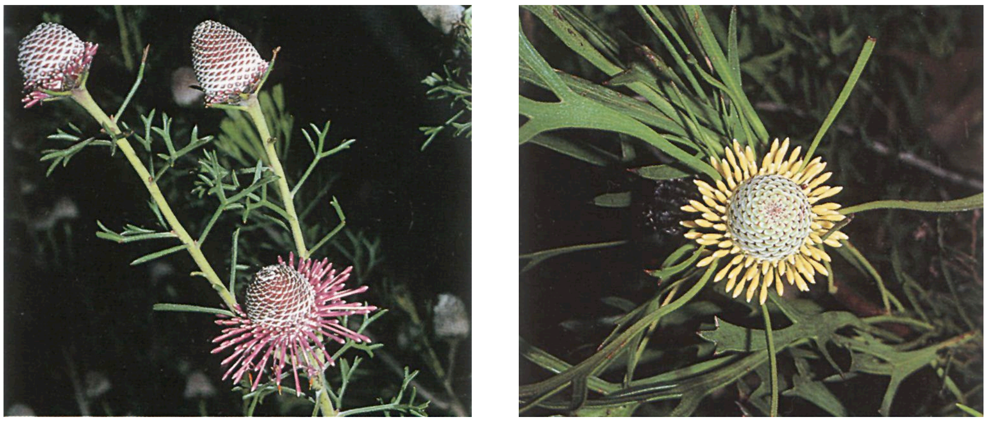
Cylindrical Phyllotaxis:

```
    #define a   137.5
    #define h   0.1
    #define r   5.0
    #define N   200
$\omega$ : A(0)
$p_1$ : $n < N \rightarrow$ [&(90) f($r$) sphere]
    f($h$) /($a$) A($n+1$)
```

Figure 7-1: Planar and Cylindrical phyllotaxis using L-systems.

An extension to the planer model is to change the element's altitude, size and orientation as a function of *n*. Changing the altitude of the elements as they are placed gives an approximation of placement over a cone or section of a cone.

While these two models have been successful in modelling a number of plant species, there exist many examples of phyllotaxis that are not described by this model (Figure 7-2 for example).



Isopogon formosus *Isopogon petiolaris*

Figure 7-2: Plants from *Isopogon* exhibiting phyllotaxis over sections of approximate prolate spheroids (from (Greig 1999)).

7.1.1 Related Work

Fowler, Prusinkiewicz and Battjes proposed a collision-based model, distributing *primordia* (undeveloped organs that eventually grow into florets) on a surface of revolution whose generating curve is defined by one or more Bézier curves (Fowler, Prusinkiewicz & Battjes 1992). Their model is "descriptive and explanatory", calculating the placement of elements using a collision-based method. In their model the radial angle, ϕ is incremented by $2\pi\varphi^{-2}$ radians, where φ is the golden ratio $\left(1+\sqrt{5}\right)/2$. The placement along the generating curve at this angle is determined by finding the minimum distance necessary to avoid collision with any previously placed element. In practice, this is computed using a binary search technique. Due to the collision-based component of the model, the method cannot directly be implemented with standard DOL-systems.

Lintermann and Deussen describe a technique for phyllotaxis over a sphere (Lintermann & Deussen 1999). They use an incremental approach based on area approximation, similar to that presented here. However, their results are limited to spheres or sub-sections of spheres only.

Prusinkiewicz et. al. present a method which operates on arbitrary surfaces of revolution (Prusinkiewicz et al. 2001), based on the model developed by Ridley (Ridley 1986). This method uses an arc-length parameterisation of planar curves, which then generate a surface of revolution. They use an incremental method to approximate the integration of the surface area as elements are placed over the surface. The technique is implemented using a derivation of Chomsky grammars and the incorporation of C-like[55] code statements into the modelling language itself.

Fleischer et. al. developed a cellular texture model that could "grow" cellular elements over pre-defined surfaces (Fleischer et al. 1995). While not specifically designed for creating phyllotactic patterns, the system gave each element (cell) a simple program (time-varying first order differential equation) that could control its placement over the surface based on (for example) simulation of chemical reaction-diffusion over the surface. The method could be considered a generalisation of the collision-based model of Fowler, Prusinkiewicz and Battjes, offering additional capabilities such as movement, adhesion and changes in size due to cell-cell interaction.

The concept of simple cells running independent programs will be further explored in Chapter 9, where the developmental cellular model, based on L-systems, is presented. Figure 7-3, *Homage to Kurt Fleischer*, replicates his "thorny spheres" model, using the cellular developmental model described in Chapter 9. The figure uses no predefined geometry or external surfaces; rather the surfaces are grown using a developmental L-system with phyllotactic placement. This image illustrates that many of the features of the cellular texture generation system can be achieved using the system described in this thesis.

[55] As in the *C* programming language (Kernighan & Ritchie 1988).

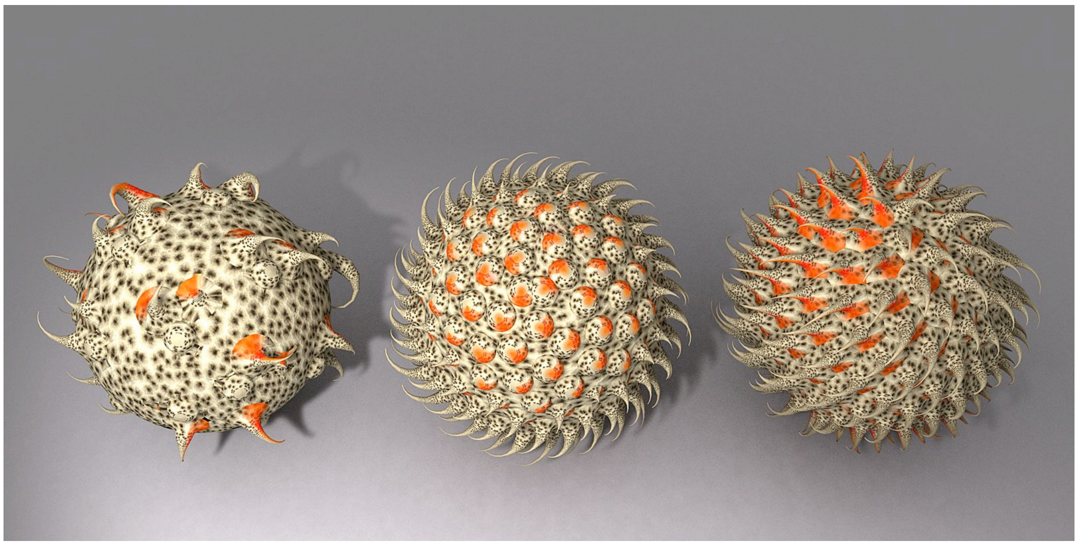


Figure 7-3: Varying Thorns in the style of Kurt Fleischer et. al. (cf. (Fleischer et al. 1995, page 247)). This image was created using the developmental cellular model detailed in Chapter 9. The placement of the thorns is controlled using the area-based phyllotactic method described here. The thorns themselves are grown using the generalised cylinder extensions described in Section 6.6.3. Simulation of reaction-diffusion-like equations determine the size, growth and direction of the bumps and thorns. Note the continuously varying thorn height and curvature on the spheres.

7.1.2 Features of the Model

The main feature of the model presented here is that it gives an analytic solution to the placement of elements over a surface of revolution. Hence, it can be incorporated into any parametric L-system description without the need for additional software or standalone components, which must be then integrated into the final model. Many of the plant organs and surfaces on which they develop can be described within the L-system using the generalised cylinder extensions introduced in Section 6.6. Hence the L-system specification is sufficient to define complete and complex three-dimensional geometric models without the need for specialised external surface modelling programs or collision-detection systems required in other approaches.

7.2 The Area-Based Phyllotactic Model

This section describes the area-based phyllotactic model in detail. Examples over a number of different surfaces are described along with some practical examples of the application of this technique.

7.2.1 Surfaces of Revolution

A surface of revolution is generated by rotating a two-dimensional curve about an axis over the interval $\left[0, 2\pi\right)$. For this discussion, it is assumed the rotation is always about the z-axis, giving the generated surface a *radial symmetry* in the xy plane (known as *azimuthal symmetry*). The *generating curve* is assumed to be a continuous, single-valued function $f_C : \Re \rightarrow \Re$, written $f_C(z)$ defined over the domain $z \in [z_{\min}, z_{\max}]$. For example, the generating curve $\left|\sqrt{r^2 - z^2}\right|$ over domain $z \in [-r, r]$ generates a sphere of radius r.

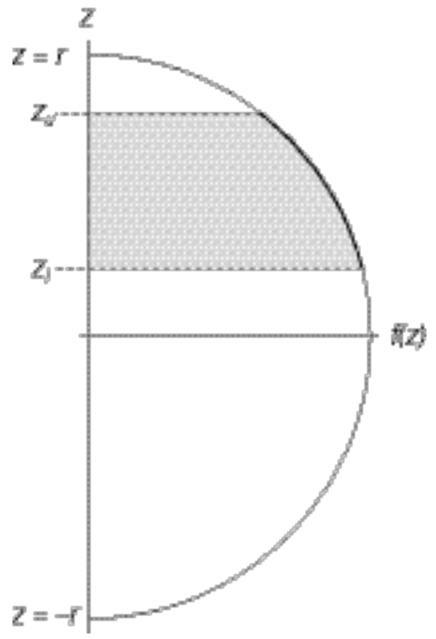


Figure 7-4: Generating curve for a sphere.

We are interested in placing geometric elements over some section of the generated surface, S, according to the golden section. The elements will be placed using turtle commands generated by the production of L-system strings as described in Chapter 1.

Due to the azimuthal symmetry of the generated surface of revolution, the area of placement can be defined in terms of the upper and lower limits $[z_l, z_u]$ of the generating curve (see Figure 7-4). The surface area of this generated surface, A_S is defined:

$$A_S = 2\pi \int_{z_l}^{z_u} f_C(z) \sqrt{1 + \left(f_C'(z)\right)^2}\, dz \tag{7.1}$$

Let the projected area[56] of element *i* to be placed on the surface be A_{e_i}, then the function

$$A_E(i) = \sum_{k=1}^{i} A_{e_k} \tag{7.2}$$

gives the total area required for *i* elements. There is no requirement for each element to have an equal projected area, however if this is the case, equation (7.2) becomes

$$A_E(i) = iA_e \tag{7.3}$$

where A_e is the projected area of the individual element. Using equation (7.1) we can calculate the maximum value of *i* needed to fill the area required.

Due to the azimuthal symmetry, we can parameterise the surface, *S*, over the axis of rotation and azimuth distance

$$S = S(\phi, h): \; 0 \le \phi < 2\pi, \, z_l \le h \le z_u \tag{7.4}$$

Note that this is different from the conventional parametric form for a surface. This is done for convenience in the calculations that follow. Figure 7-5 shows the parameters for a sphere.

[56] The projected area can be calculated by orientating the element so that the *xy* plane corresponds to the tangential plane of the surface on which the element is to be placed. The projected area is then defined $\iint_R dxdy$, where *R* is the region over which the element is defined.

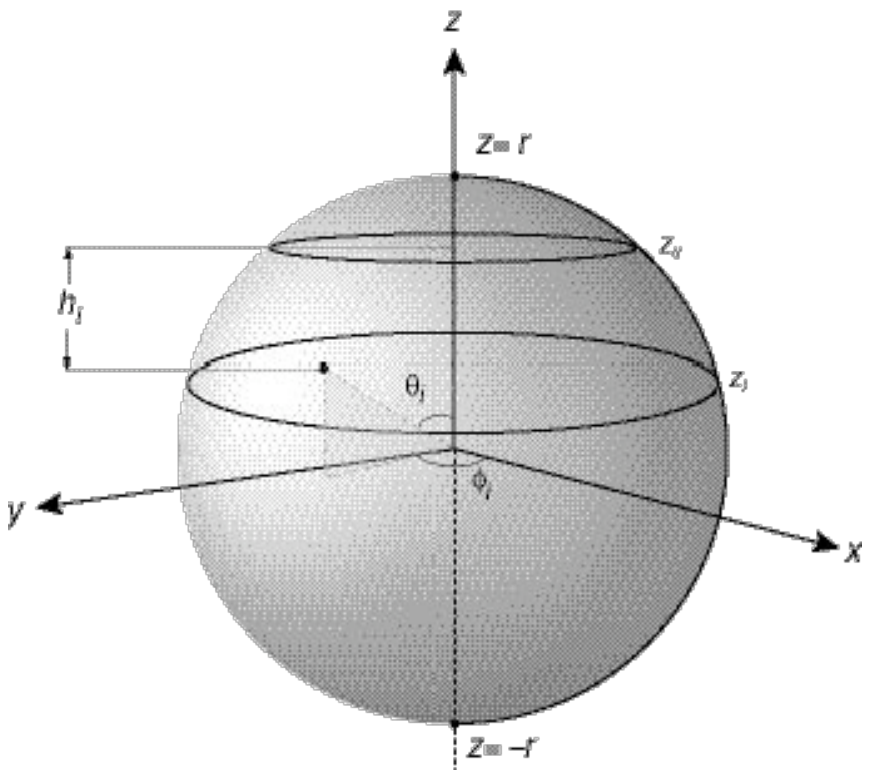


Figure 7-5: Sphere generated as a surface of revolution.

As each element, i, is placed on the surface, the parameters ϕ_i and h_i are required to determine the location of the element on the surface. The radial angle, ϕ_i is distributed

$$\begin{aligned}\phi_i &= i\ d\phi \\ &= i\frac{2\pi(\varphi-1)}{\varphi} \\ &= \left(3-\sqrt{5}\right)\pi\ i\end{aligned} \tag{7.5}$$

where φ is the golden ratio (≈ 1.61803). The numerical value of ϕ_i equates to approximately 137.508°. Using equation (7.1), the surface area of the surface of revolution at h_i can be written:

$$A_S(h_i) = 2\pi\int_{z_u-h_i}^{z_u} f_C(z)\sqrt{1+\left(f_C{'}(z)\right)^2}\,dz \tag{7.6}$$

To ensure correct placement, it is required that

$$A_E(i) \equiv A_S(h_i) \tag{7.7}$$

hence

$$h_i = A_S^{-1}\left(A_E(i)\right) \tag{7.8}$$

Depending on the nature of the generating curve, equation (7.8) may not have an algebraic solution, as finding an inverse function may be impossible. In this case, numerical methods may be employed. Alternatively, another solution is to use an approximating polynomial of limited order (< 4), to which an inverse can usually be found.

The position to place the element on the surface, S, can be expressed in terms of the turtle commands:

$$/(\phi_i)\ \mathtt{f}(z_u - h_i)\ \ \mathtt{\hat{}(90)}\ \ \mathtt{f}(f_C(h_i)) \tag{7.9}$$

The interpretation of these turtle commands was detailed in Chapter 1.

7.2.1.1 Orientation of the Element

The positional information described above provides a location to place elements on the surface of revolution. In addition, a specific orientation may also be required. If this is the case, the element needs to be aligned to the surface normal of the surface of revolution, $\mathbf{n}_S$.

For surfaces with a parametric representation, the normal is the orthogonal vector to the tangent plane. That is, for a surface $\mathbf{r}(u,v)$ the normal vector is

$$\begin{aligned} \mathbf{n}_\mathrm{r} &= \frac{\partial \mathbf{r}}{\partial u} \times \frac{\partial \mathbf{r}}{\partial v} \\ &= \mathbf{r}_u \times \mathbf{r}_v \end{aligned} \tag{7.10}$$

In the case of a surface of revolution, the normal vector can be found by differentiating the generating curve and then rotating this vector to the radial location in ϕ. In practice, to align the element to the surface normal requires orienting the turtle's co-ordinate reference frame to the triplet $\begin{bmatrix}\mathbf{r}_u & \mathbf{r}_v & \mathbf{n}_r\end{bmatrix}$. Following the turtle commands specified in (7.9), the turtle requires a rotation, ρ, based on the normal to the generating curve:

$$\rho = \tan^{-1}\left(f_C{}'(h)\right) \tag{7.11}$$

Assuming the turtle is at the correct position on the surface with the heading vector aligned with the z axis, a turtle command of $\mathtt{\hat{}(\rho)}$ will align the turtle to the surface normal.

7.2.2 Example: Sphere

A sphere of radius *r* can be created by revolving the curve

$$f_C(z) = f_{sphere}(z) = \left|\sqrt{r^2 - z^2}\right|, z \in \left[-r, r\right] \tag{7.12}$$

about the z axis over $[0, 2\pi)$ (Figure 7-5). By equation (7.6) the area of the surface defined from the boundaries indicated in Figure 7-5 is

$$\begin{aligned} A_{sphere}(h_i) &= 2\pi \int_{z_u - h_i}^{z_u} \sqrt{r^2 - z^2} \sqrt{1 + \left(-\frac{z}{\sqrt{r^2 - z^2}}\right)^2} dz \\ &= 2\pi \int_{z_u - h_i}^{z_u} r\, dz \\ &= 2\pi r h_i \end{aligned} \tag{7.13}$$

Using the equivalence relation of equation (7.7)

$$h_i = \frac{A_E(i)}{2\pi r} \tag{7.14}$$

If the elements all have the same projected area then

$$h_i = \frac{iA_e}{2\pi r} \tag{7.15}$$

h_i can be expressed in terms of θ_i, the azimuth angle between the positive z axis and the vector formed from the centre of the sphere to the position of element *i* on the sphere

$$\theta_i = \cos^{-1}\left(\cos\theta_u - \frac{iA_e}{2\pi r^2}\right) \tag{7.16}$$

where θ_u is the value of θ at z_u (i.e. h_0). Note that the angle θ is equivalent to a parameter when the sphere is represented as a parametric surface, i.e., $S(\theta, \phi)$.

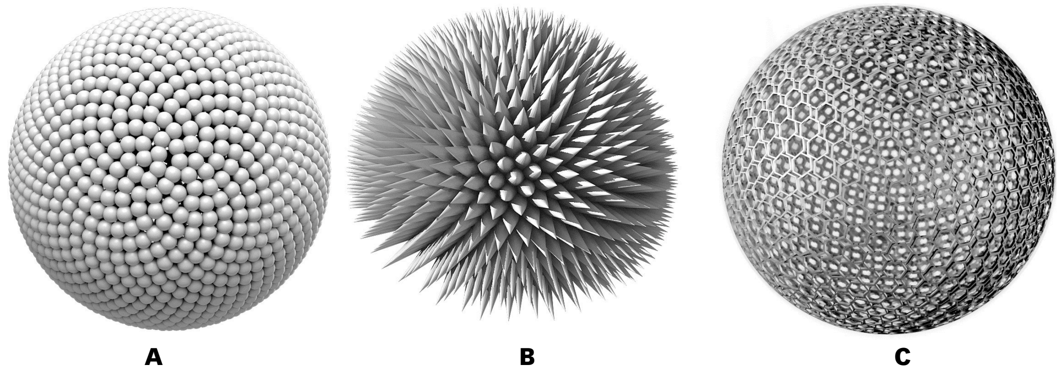


Figure 7-6: Phyllotaxis of various geometric elements over the surface of a sphere. Elements placed are a sphere (**A**), Cone (**B**) and hexagon (**C**).

Figure 7-6 shows some examples. For the examples shown in this figure, each element is of constant projected area (i.e. all elements are the same size). Figure 7-6C is reminiscent of some of the skeletal forms of *Radiolaria*, studied by Thompson (Thompson 1961, pages 151–159).[57]

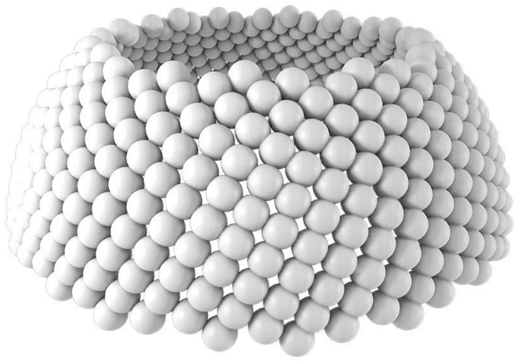


Figure 7-7: Elements arranged on a sphere from $\frac{\pi}{6} \le \theta \le \frac{\pi}{2}$.

Figure 7-7 shows an example where $|z_l| < r$ and $|z_u| < r$, thus covering only a section of the entire sphere.

Elements of constant area do not correctly simulate the growth of plant elements such as florets (Jean 1982). Figure 7-8 shows an example where the elements placed on the sphere do not have a constant area. In the case of this

[57] Thompson was motivated in his study by the work of Ernst Haeckel, who drew *Aulonia hexagona* using hexagons. Thompson correctly stated that "no system of hexagons can enclose space" (Thompson 1961, page 157). This was also acknowledged by Haeckel, who observed that a few square or pentagonal facets are found among the hexagons. An additional twelve pentagons are needed to correctly form a closed space, as was demonstrated subsequently in the architectural constructions of R. Buckminster Fuller.

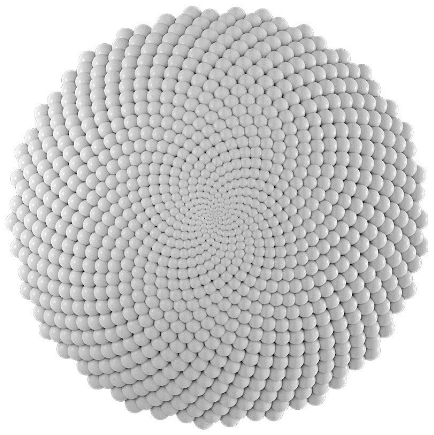
figure, the element has an exponential growth function, whereby the radius of each element (a sphere in the case of Figure 7-8A) grows according to the function

$$r = c_1\left(1 - e^{-c_2 k}\right) \tag{7.17}$$

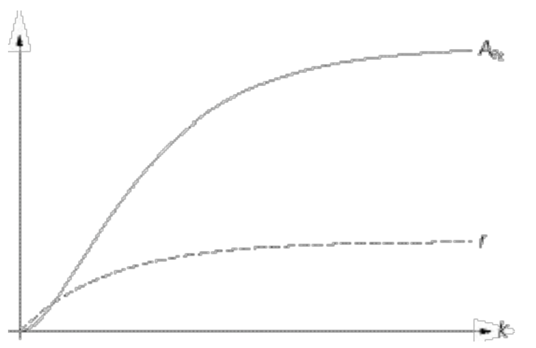
where c_1 and c_2 are constants. This function is designed to simulate the growth of florets from the centre of a floral head, the youngest elements appearing at the apex of the sphere. For elements with a circular projected area the value of A_{e_k} from equation (7.2) is given by

$$\begin{aligned} A_{e_k} &= \pi r^2 \\ &= c_1\pi\left(1 - e^{-c_2 k}\right)^2 \end{aligned} \tag{7.18}$$

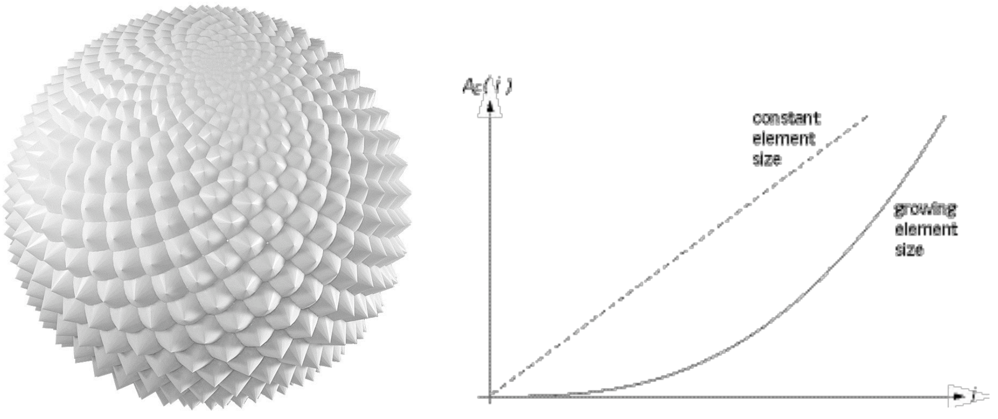
The figure shows a graph of this function and the cumulative area function $A_E(i)$ (figure 7-8B and D). Figure 7-8D also shows the value of this function for elements of constant area for comparison.



A. Spheres of increasing size placed over a larger sphere. The growth of the individual sphere's radius is as specified in equation (7.18).



B. Radius (*r*) and projected area (A_{e_k}) of individual elements increase as they are placed on the surface.



C. Conical elements of increasing size placed over the surface of a sphere.

D. Cumulative area of elements as their number increases, shown for elements of constant projected area (dashed line) and increasing projected area (solid line) as shown in **C**.

Figure 7-8: Elements of differing size placed over a sphere.

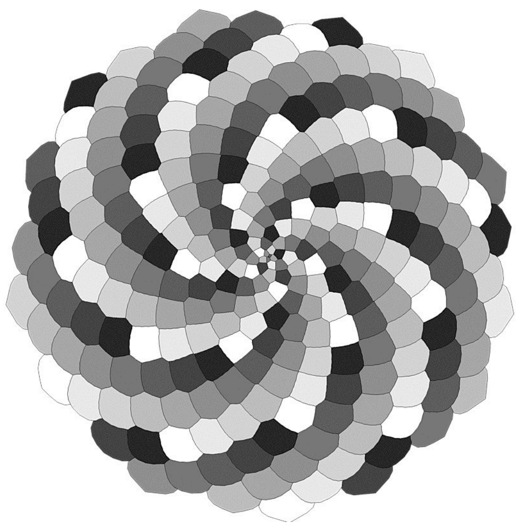


Figure 7-9: Illustrates the growth and placement of the elements. A ten step sequence of shades of grey (darkest to lightest) shows the relative age of elements as they are placed on the surface.

7.2.3 Example: Oblate and Prolate Spheroids

Using the methodology described in Section 7.2.1, as for the sphere we can define a surface of revolution that describes a spheroid. Two types of spheroids are described here, *oblate* or "squashed" and *prolate* or "stretched" spheroids (Figure 7-10). Both these shapes can be described by rotating an ellipse about its minor (for oblate) or major (for prolate) axis. The motivation for using such shapes is the observed profiles of plant organs that were previously approximated by cylinders (cf. Figure 7-2). The cylindrical approximation leads to inconsistencies at the base and top.

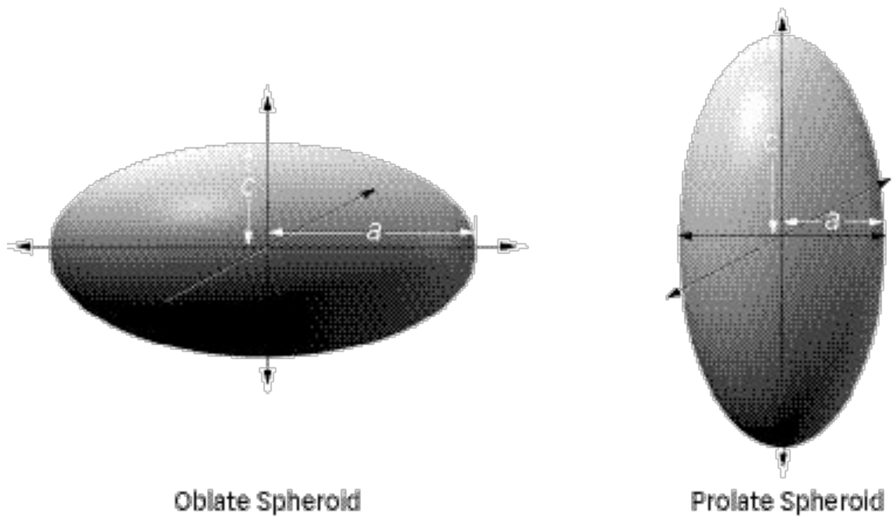


Figure 7-10: Oblate and Prolate spheroids.

The implicit form of these spheroids is

$$\frac{x^2+y^2}{a^2}+\frac{z^2}{c^2}-1=0 \tag{7.19}$$

where a is the *equatorial radius* and c is the *polar radius*. For an oblate spheroid $a > c$, for a prolate spheroid $a < c$.

Proceeding as for the case of a sphere (Section 7.2.2), the spheroid can be formed by rotating the elliptical curve

$$f_C(z)=f_{ellipse}(z)=a\left|\sqrt{1-\left(\frac{z}{c}\right)^2}\right| \tag{7.20}$$

about the z axis over $[0,2\pi)$ (Figure 7-5). By equation (7.6) the area of the surface defined from the boundaries indicated in Figure 7-5 is

$$\begin{aligned} A_{spheroid}(h_i) &= 2\pi\int_{z_u-h_i}^{z_u} f_{ellipse}(z)\sqrt{1+\left(f'_{ellipse}(z)\right)^2}\,dz \\ &= 2\pi a\int_{z_u-h_i}^{z_u}\sqrt{1+\frac{(a-c)(a+c)z^2}{c^4}}\,dz \end{aligned} \tag{7.21}$$

Evaluating this definite integral gives:

$$A_{spheroid}(h_i) = \pi a \left(z_u \sqrt{1 + z_u^2 \left(\frac{a^2 - c^2}{c^4} \right)} - (z_u - h_i) \sqrt{1 + (z_u - h_i)^2 \left(\frac{a^2 - c^2}{c^4} \right)} + \frac{c^2 \left(\sinh^{-1}\left(z_u \frac{\sqrt{a^2 - c^2}}{c^2} \right) - \sinh^{-1}\left((z_u - h_i) \frac{\sqrt{a^2 - c^2}}{c^2} \right) \right)}{\sqrt{a^2 - c^2}} \right) \tag{7.22}$$

In the case of a prolate spheroid, $a < c$ and so $\sqrt{a^2 - c^2}$ will have a complex root. However, due to the nature of the hyperbolic sine function, the equation evaluates to a real solution that does not involve complex numbers. Figure 7-11 shows a graph of this equation for both oblate and prolate spheroids with equatorial to polar radius ratios of 0.75:1 and 1.25:1 respectively. Finding the inverse function needed to satisfy equation (7.8) is not possible analytically in the general case. However, it is possible to approximate the function using a polynomial, in which case an inverse can be found. Therefore:

$$h_i = P^{-1}_{spheroid}\left(A_E(i)\right) \tag{7.23}$$

where $P_{spheroid} \approx A_{spheroid}$ is the polynomial approximation to equation (7.22). A specialised function that returns the value of h_i when supplied the parameters a, c, z_u and $A_E(i)$, can be made available to the generating L-system to avoid the complex parameter manipulation necessary to incorporate this solution into the L-system rules.

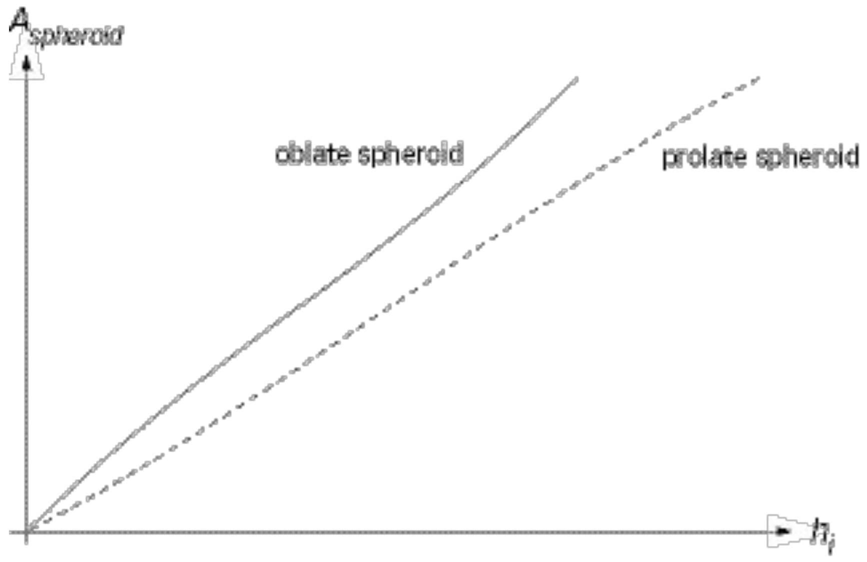


Figure 7-11: Surface area of oblate (solid line) and prolate (dashed line) spheroids as a function of distance from the polar radius along the *z* axis.

Figure 7-12 shows simple conical elements placed over oblate and prolate spheroids using the method described here.

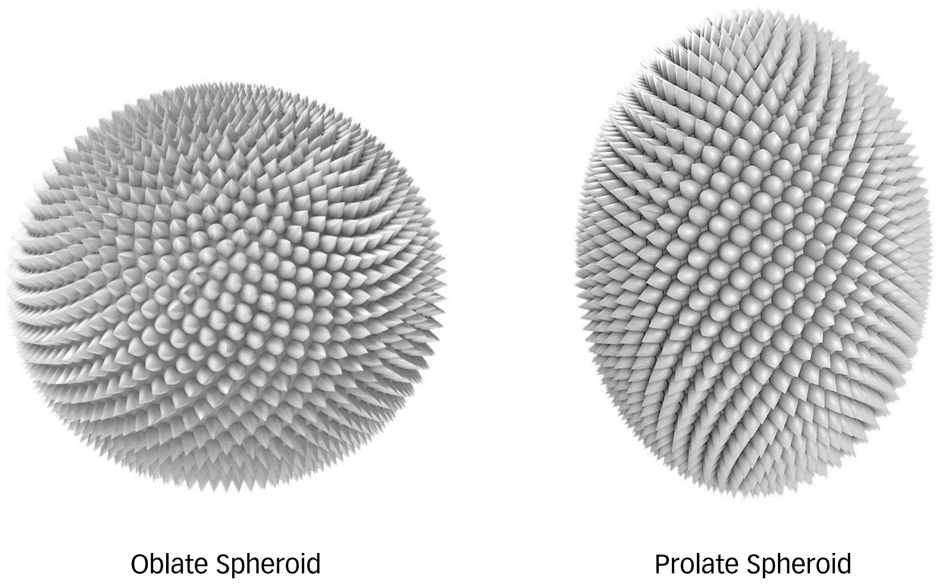


Figure 7-12: Oblate and Prolate spheroids with phyllotactic placement of elements over their surfaces.

The use of such spheroids finds application in the realistic visual modelling of a variety of different plants. For example, Figure 7-13 and Figure 7-14 illustrate some possibilities. Figure 7-13 is typical of many members of the *Myrtaceae* family. The cluster of stamens is modelled using generalised cylinders, with length, size and growth direction subject to modulation using a normal (Gaussian) distribution.

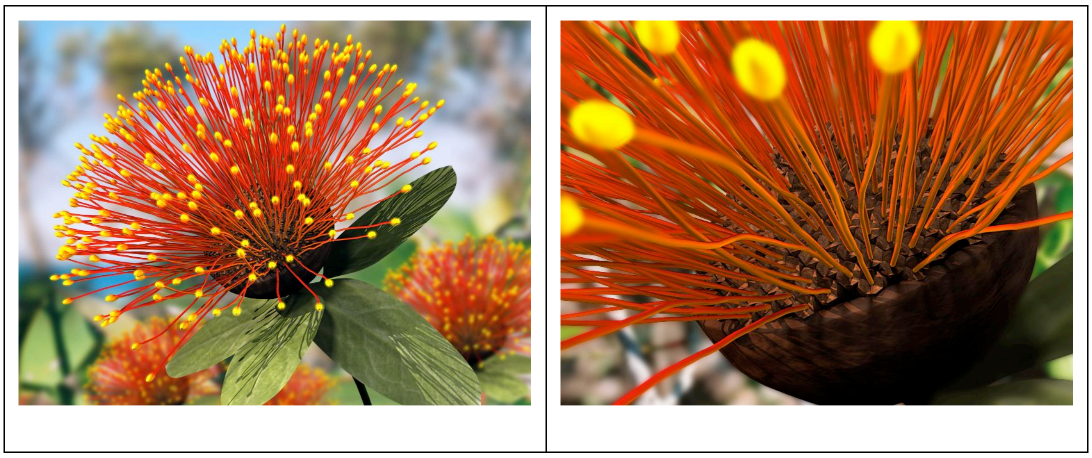


Figure 7-13: Plant model created using the system described in this thesis. No external geometry was used to create this model. Textures were defined procedurally or sourced from real specimens. Variation in the length, size and placement is achieved using the statistical functions detailed in Section 6.8. The image on the right shows a close-up revealing the model detail.

High-resolution colour plates of these images can be found in Appendix A.

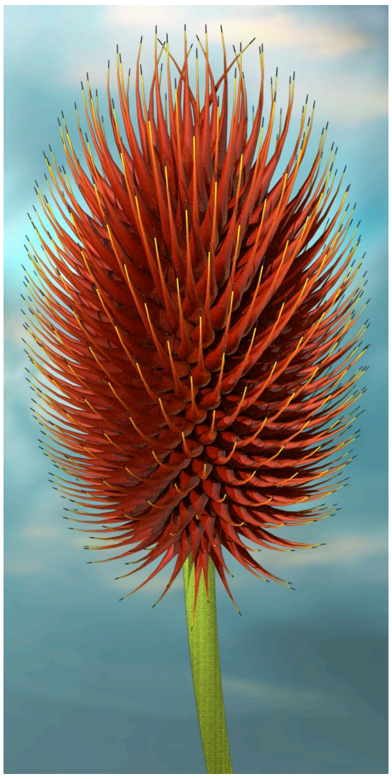


Figure 7-14: Model created by placing thorn-like elements over a prolate spheroid. The rate of curvature varies as the elements are placed using the `smoothstep` function.

7.3 Stochastic Functions

Even within a species, no two individuals are identical. The models presented thus far represent idealisations of various observed phenomena. For example, the examples of phyllotaxis presented earlier in this chapter represent the ideal form, with exact spacings and identical florets placed over idealised surfaces. While such models may capture the mathematical essence of specific phenomena, these ideal forms are never observed in nature. There are always small deviations, damage, mistakes and variation that make each individual unique.

It is possible to capture a sense of this variation using one, or a combination of the stochastic and noise based functions presented in Section 6.8.2. This section will illustrate some practical examples for which these functions may be used.

7.3.1 A Simple Grass Model

As a simple example, we consider the challenge of modelling a grass field using L-systems. A basic model features long, thin strips of grass, with possible bifurcation of stalks based on width. Grasses may be in different states (juvenile, flowering, seeding). A basic L-system model consists of a number of segments, connected to form a generalised cylinder. E.g.

$$
\begin{array}{lll}
\#\text{define } k_b & \frac{\pi}{20} & // \text{ bend} \\
\#\text{define } k_r & 5 & // \text{ radius} \\
\#\text{define } k_l & 120 & // \text{ segment length} \\
\#\text{define } k_s & 8 & // \text{ number of segments} \\
\#\text{define } \delta_b & 1.1 & // \text{ bend delta} \\
\#\text{define } \delta_r & 0.8 & // \text{ radius delta} \\
\#\text{define } \delta_l & 0.8 & // \text{ segment length delta} \\
\end{array}
$$
$$
\begin{array}{l}
\omega : c(1)\, g(0, k_b, k_r, k_l) \\
p_1 : g(n,b,r,l) : n < k_s : \rightarrow \&(b)\,!(r)\, C(l)\, g(n+1, \delta_b k_b, \delta_r k_r, \delta_l k_l)
\end{array}
\tag{7.24}
$$

The constants k_b, k_r and k_l control segment bending, radius, and length respectively, while k_s specifies the number of segments that make up the stalk. The other constants δ_b, δ_r and δ_l control the change in bend, radius and length as each segment is produced. An example single stalk produced by this L-system is shown in Figure 7-15. By defining a more accurate profile based on the studies by Wainwright (Wainwright 1988, Chapter 2), a more realistic model is obtained. This can be achieved using the profile definition commands defined in Section 6.6.2.

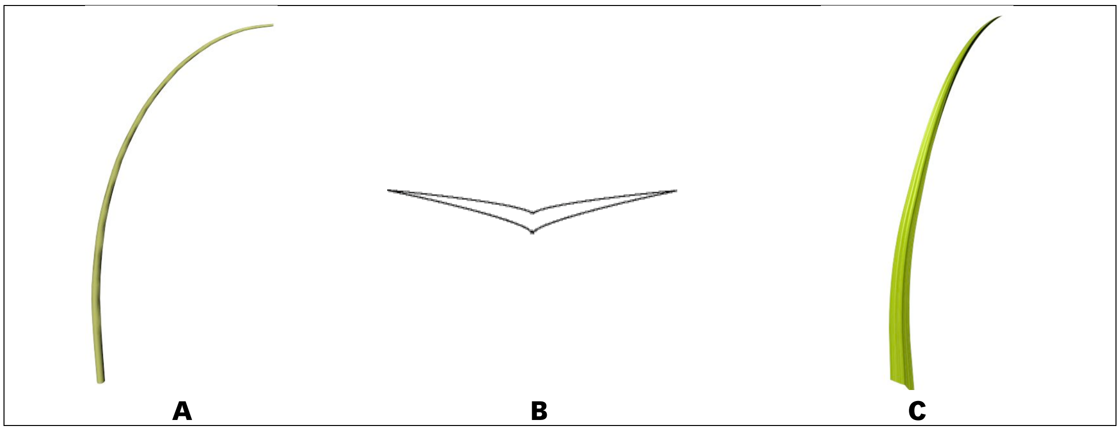


Figure 7-15: Single grass stalk produced by the L-system (7.24) using generalised cylinders with a circular profile (**A**). Using the profile (**B**) results in a more physically accurate model (**C**).

The basic model remains unchanged, but such a profile gives the organ an uneven relation between twisting (low *torsional stiffness*) and bending (high *longitudinal stiffness*), giving the organ the ability to blow readily in the wind without breaking off.

It is a relatively simple process to produce many of these stalks to make a more complex model. However, each stalk will be identical to all the others. A more realistic model can be obtained by changing the δ constants to functions:

$$\begin{aligned} \delta_b &= \mathtt{gaussian}(1.1, 0.1) \\ \delta_r &= \mathtt{gaussian}(0.8, 0.067) \\ \delta_l &= \mathtt{gaussian}(0.8, 0.1) \end{aligned} \tag{7.25}$$

Now, changes between segments will have a normal distribution with mean values the same as for the "ideal" stalk of Figure 7-15. An additional function, based on the noise function can control segment twisting; simulating stalks blowing in the wind. Changing the variance parameter controls the variation in how much instanced models differ from the ideal model. Figure 7-16 shows a sequence of models generated using the distributions of (7.25).

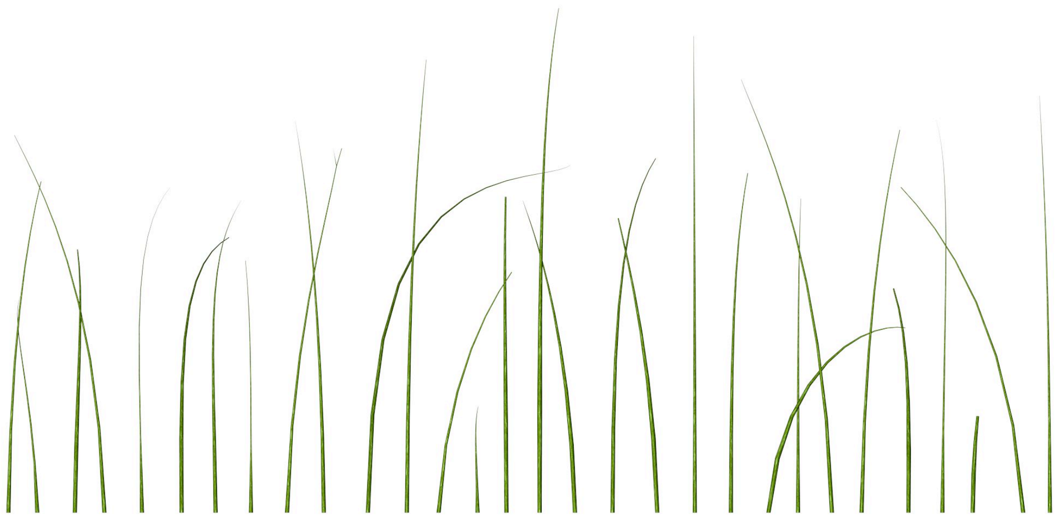


Figure 7-16: A number of grass stalks generated by normally distributing the δ_b, δ_r and δ_l constants.

A more complex addition to this simple model adds a normally distributed age parameter, which triggers the development of flowers. The production p_1 of L-system (7.24) thus becomes:

$$\begin{aligned}
&\#\,\text{define}\ \delta_\tau \quad \frac{\mu}{e^2}\\
&\Omega = \texttt{gaussian}(k_s, \mu)\\
&\tau(s) = \texttt{poisson}_s(\Omega) \qquad\qquad\qquad\qquad\qquad\qquad\qquad (7.26)\\
&p_1 : g(n,b,r,l) : n < \tau(g) : \rightarrow \&(b)\,!(r)\ C(l)\ g(n+\delta_\tau, \delta_b k_b, \delta_r k_r, \delta_l k_l)\\
&p_2 : g(n,b,r,l) : n > \Omega : \rightarrow \varnothing\\
&p_2 : g(n,b,r,l) : n > \tau(g) : \rightarrow \mathit{flower}(r,l)
\end{aligned}$$

Here the function $\tau(s)$ is a pseudorandom number with Poisson distribution, keyed to the symbol *s*. Keying this function to a particular symbol ensures consistent values are returned for the same instance of a particular symbol (in this case, *g*). The mean of this distribution is normally distributed about k_s — the number of segments in the "ideal" stalk. As the stalk grows, its "age", represented by *n* is checked to see if it exceeds τ, if so, the production proceeds to draw a flower (the flower drawing production is not shown). By changing the variance, μ, the number of stalks which flower can be changed. A similar scheme can be used to trigger the development of seeds and other stalks, based in the phyllotactic model given in Section 7.2.

7.3.2 Distribution over Surfaces

We have now developed a simple model with variation and different growth stages. However, this currently only defines a single individual. A large population of individuals can be distributed over a surface. For this example, we will use a Poisson distribution, as this is a commonly observed natural distribution pattern (Ripley 1977). Models that are more detailed could simulate variations in landscape, such as environmental conditions, hydration, or soil nutrient gradients (Deussen et al. 1998). These models require external software to model the distribution of species so they cannot be directly implemented with L-systems.[58]

The *point pattern* generation model described here is sufficiently simple as to be directly implemented as part of the L-system specification. The basic goal is a random distribution of plants over a surface with a specified mean density. An added requirement that a minimum distance constraint for nearest neighbours is not violated. The minimum distance constraint specifies the minimum distance allowed between any individual plants. Similar techniques

[58] Extensions to L-systems have been proposed that take environmental modelling into account, e.g. (Mech & Prusinkiewicz 1996).

have been used for point sampling in computer graphics (Cook, R. L. 1986). To generate the model we begin with a two-dimensional uniform grid spaced to satisfy the density requirements of the particular model. In a more complex version of the model, grid spacing varies dynamically based on varying density constraints. Points on the grid are then randomly shifted to simulate a distribution of elements over the surface.

The following L-system captures the process:

$$
\begin{array}{ll}
\#\,\text{define } N_x \quad 10 & // \text{ points in } x \\
\#\,\text{define } N_y \quad 10 & // \text{ points in } y \\
\#\,\text{define } \Delta \quad 100 & // \text{ spacing (inverse density)} \\
\delta = \dfrac{\pi}{2} & // \text{ default turtle turn angle} \\
\omega : col(0) & \\
p_1 : col(j) : j < N_y \rightarrow [+\, row(0, j)]\; f(\Delta)\; col(j+1) & \\
p_2 : row(i, j) : i < N_y \rightarrow [-\, f(\zeta) + f(\zeta)\ \wedge\ element(i, j)\ ]\; f(\Delta)\; row(i+1, j) &
\end{array}
\tag{7.27}
$$

where $\zeta \in [0, \Delta]$ is a function that controls the shifting or *jitter* of the position of elements. The symbol *element* represents the placement of an element at the current turtle position and orientation.

Figure 7-17 shows how the use of different functions for ζ changes the distribution of elements (represented by dots in this figure). To satisfy the minimum distance constraint, Δ and the range returned by the function ζ must be chosen appropriately. The choice of function controls the distribution of plants over the surface. Note that in this case the distribution is over a flat two-dimensional surface, a simple extension would provide distribution over a terrain model, either procedurally generated (Musgrave, Kolb & Mace 1989) or sampled from real data (House et al. 1998). This is described in the next section.

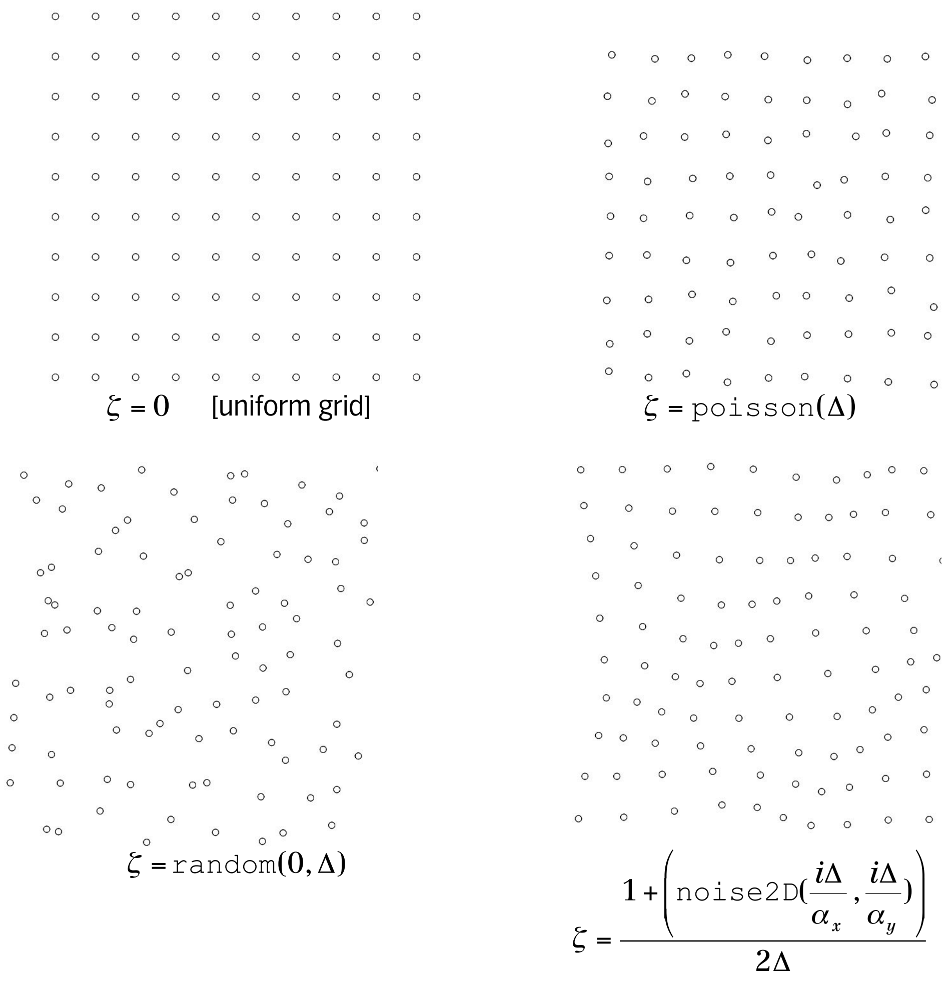


Figure 7-17: Patterns produced by assigning different functions to ζ created using the L-system defined in (7.27).

7.3.3 Terrain Generation and Distribution

"Fractal" methods of terrain generation have been extensively studied in computer graphics, (Carpenter 1980; Fournier, Fussell & Carpenter 1982; Voss & Clarke 1975) being early examples, good overviews can be found in (Ebert et al. 1994; Musgrave, Kolb & Mace 1989). In this example, a terrain model will be generated using L-systems and based around the `noise`, `fractalSum` and `turbulence` functions described in Section 6.8.2. Somewhat similar techniques have been proposed by Prusinkiewicz and Hammel who used midpoint-displacement methods to generate mountains, along with an integrated squig-curve model for rivers (Prusinkiewicz & Hammel 1993). They generated their model using context-sensitive re-writing system in order to share information between edges of the subdivided triangles.

Two methods of procedural terrain construction using L-systems have been explored: a *recursive subdivision* method and *uniform subdivision* method. Both models rely only on a function, $\lambda : \Re^2 \rightarrow \Re$, which can return a *height-field* value. Height-fields give altitude values at some two-dimensional position (x, y). Examples of procedural functions suitable for terrain generation can be found in (Musgrave 1994). Alternatively, actual sampled data or data drawn using a form of paint program can be used. One clear disadvantage of the height-field definition of terrain heights is that only one altitude can exist for any given two-dimensional position. This means features such as caves, non-vertical holes, and overhanging cliffs cannot be described using this technique.

7.3.3.1 Recursive subdivision

The basic method is to recursively sub-divide a regular polygon or *tile*, until it reaches the desired resolution. Subdivision stops when the height difference between points being subdivided falls below some threshold (σ) or until some maximum limit in subdivision is reached.

To illustrate the principle let us assume a two-dimensional “mountain” cross section, drawn with a line. This can be easily extended to draw a surface. The function $\lambda(\mathbf{p})$ returns the height of the terrain at the point **p**. The constant L determines the maximum number of subdivisions.

$$\begin{array}{l}
\quad \#\text{define} \quad L \qquad 8 \ \ // \text{ maximum recursive depth} \\
\delta = \dfrac{\pi}{2} \\
\omega : bP \; V(\mathbf{p}_{\min}) \; sub(\mathbf{p}_{\min}, \mathbf{p}_{\max}, 0) \; V(\mathbf{p}_{\max}) \; eP \\
p_1 : sub(\mathbf{p}_a, \mathbf{p}_b, i) : i < L \;\&\&\; \left|\lambda(\mathbf{p}_b) - \lambda(\mathbf{p}_a)\right| > \sigma \rightarrow \\
\quad [\; sub(\mathbf{p}_a, \texttt{linear}(\mathbf{p}_a, \mathbf{p}_b, 0.5), i+1) \;] \; f\left(\left|\dfrac{\mathbf{p}_b - \mathbf{p}_a}{2}\right|\right) \\
\quad V(\lambda(\texttt{linear}(\mathbf{p}_a, \mathbf{p}_b, 0.5))) \\
\quad [\; sub(\texttt{linear}(\mathbf{p}_a, \mathbf{p}_b, 0.5), \mathbf{p}_b, i+1) \;] \; f\left(\left|\dfrac{\mathbf{p}_b - \mathbf{p}_a}{2}\right|\right) \\
p_2 : sub(\mathbf{p}_a, \mathbf{p}_b) : \left|\lambda(\mathbf{p}_b) - \lambda(\mathbf{p}_a)\right| \le \sigma \rightarrow \varepsilon \\
p_3 : V(\mathbf{p}) \rightarrow [\; \wedge \; f(\lambda(\mathbf{p})) \cdot \;]
\end{array} \tag{7.28}$$

Figure 7-18 shows the results of running this L-system with $\lambda(\mathbf{p}) = \alpha_m\ \texttt{noise}(\alpha_s \mathbf{p})$, where α_m is a constant scaling factor and α_s a constant determining the average frequency per unit area.

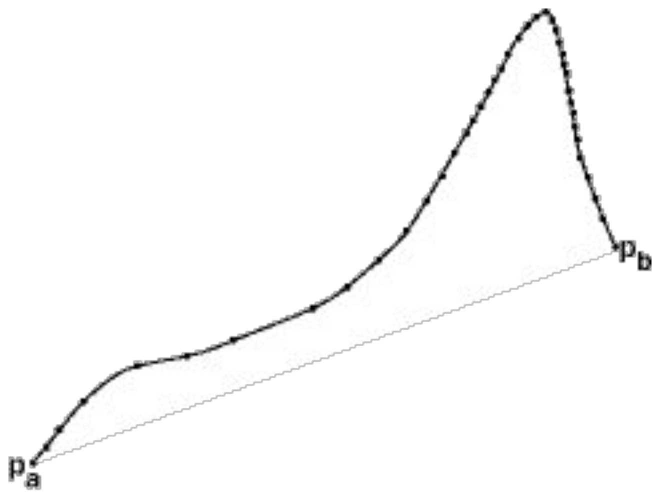


Figure 7-18: Output of L-system (7.28)

To convert to a surface representation, the subdivision productions p_1 and p_2 are applied to a square (defined by four points) rather than a line (defined by two points).

Ideally, the subdivision method keeps subdividing until a subdivision would have minimal effect on the resultant surface. However, in practice, examining differences between samples (the conditional $|\lambda(\mathbf{p}_b) - \lambda(\mathbf{p}_a)| > \sigma$ in production p_1) only works accurately on continuous, monotonically increasing functions over $[\mathbf{p}_a, \mathbf{p}_b]$. This problem can be circumvented with some explicit knowledge of the `noise` function (specifically that it is usually generated as a Hermite polynomial between points on a lattice (Ebert et al. 1994)). The subdivision method can also be used to create fractal surfaces using the midpoint displacement technique (Fournier, Fussell & Carpenter 1982), using the function $\lambda(\mathbf{p}) = \texttt{random}_{\mathbf{p}}(-\alpha_m, \alpha_m)$. Note that this function must be keyed to a particular location in order to create a continuous surface.

7.3.3.2 Uniform Subdivision

The uniform subdivision method simply increments over the area required for the terrain, moving to the specified height at each step and creating triangles.

$$
\begin{array}{l}
\quad \#\text{define} \quad N_\text{i} \qquad 100 \ // \text{subdivisions in } x \\
\quad \#\text{define} \quad N_\text{j} \qquad 100 \ // \text{subdivisions in } y \\
\delta = \dfrac{\pi}{2} \\
\omega : col(0) \\
p_1 : col(i) : i < N_i \rightarrow [\, + \, row(0,i) \,] \; f(\Delta) \; col(i+1) \\
p_2 : row(j,i) : j < N_j \rightarrow bP \; V((\Delta i, \Delta j)) \; f(\Delta) \\
\quad [\; V((\Delta(i + \texttt{if}(\texttt{odd}(j),0,1)), \Delta(j + \texttt{if}(\texttt{odd}(j),1,0)))) \\
\quad\quad - (\delta \, \texttt{if}(\texttt{odd}(j),-1,1)) \; f(\Delta) \; V((\Delta(i - \texttt{if}(\texttt{odd}(j),1,0)), \Delta j + 1)) \;] \; eP \\
p_4 : V(\mathbf{p}) \rightarrow [\; \wedge \; f(\lambda_\mathbf{p}(\mathbf{p})) \cdot \;]
\end{array}
\tag{7.29}
$$

The constant Δ specifies the sampling interval and the function λ_p is the height field function. Figure 7-19 shows the resultant surface with $\lambda_\text{p}(\mathbf{p}) = \alpha_1(\alpha_n \texttt{noise}_\text{p}(\mathbf{p}) + \alpha_f \texttt{fractalSum}_\text{p}(\beta_f \mathbf{p}, \kappa))$, where α_1, α_n, α_f and β_f are all scaling constants. κ represents the number of octaves of summed noise. For the image shown in the figure $\alpha_1 = 100$, $\alpha_n = 0.7$, $\alpha_f = 0.3$, $\beta_f = 2$ and $\kappa = 3$. This gives a combination of rolling slopes (due to the contribution of the `noise` function), with a subtle amount of detail from the `fractalSum` function.

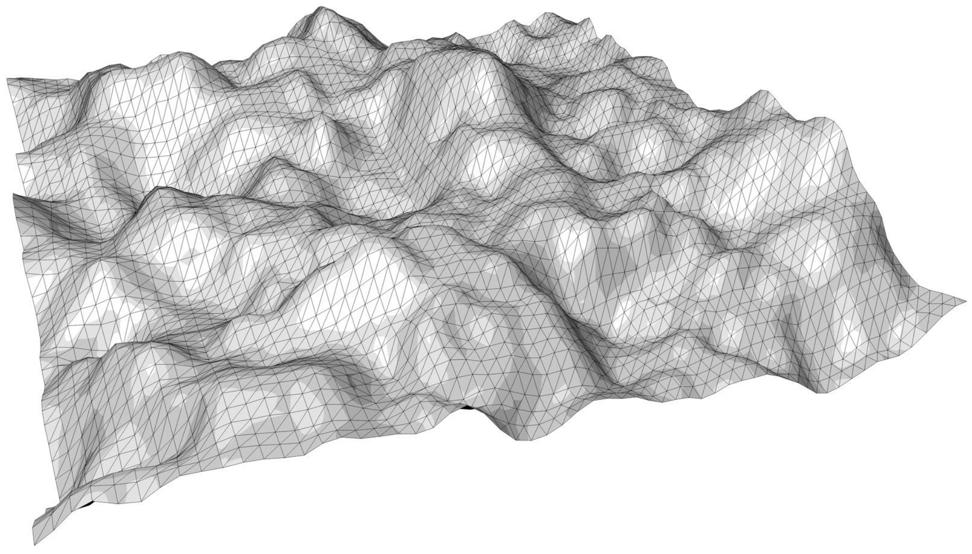


Figure 7-19: Fractal terrain generated using the uniform subdivision method of L-system 7.29.

A “water” plane can be added to separate land from water (Figure 7-20). The plane is arbitrarily defined on the $z = 0$ plane, meaning any terrain with a negative z value is underwater. Due to the characteristics of the height-field function used, this gives approximately a 50% of land to water ratio. Lowering or raising the water plane increases or decreases this ratio respectively. The choice of the water height will become important for the distribution model described in the next section.

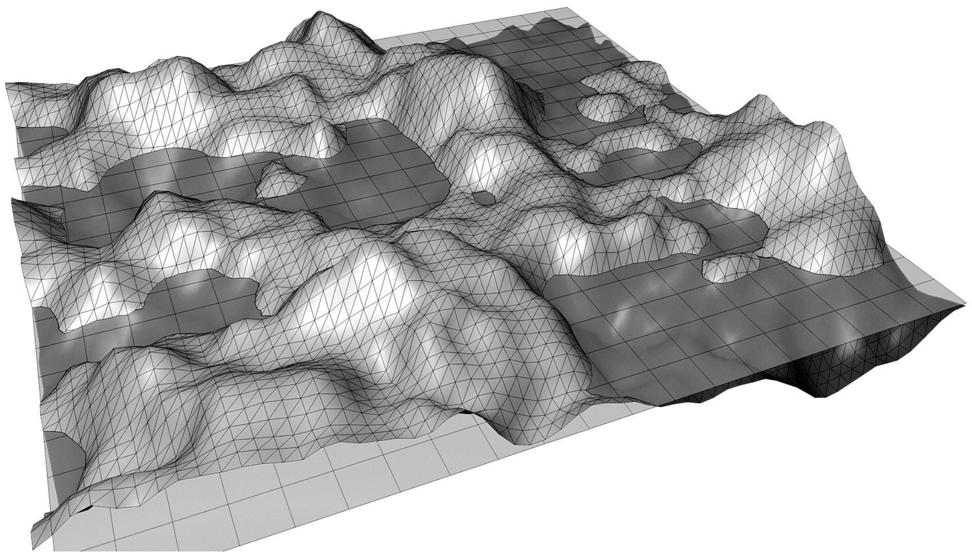


Figure 7-20: Landscape with plane at *z=0* added, representing the water line. Geometry below the plane is underwater, above, land.

7.3.4 Element Placement over Terrain

Elements need to be placed over the terrain using the distribution methods outlined in Section 7.3.2. With the addition of a height-field, placement becomes more complex than for a flat surface if we wish to generate at specified density per unit area. In addition, different species may be distributed over different altitudes. Some species may require well-drained soil on steep slopes, others may favour close proximity to water catchment areas such as streams and lakes.

The simple model presented here allows the distribution of species (or variation of species generated) based on the altitude. Each species is normally distributed with different mean and variance values for altitude. Distribution may also be clamped to prevent any plants that grow at low altitudes from growing underwater. Figure 7-21 shows such a distribution for a number of different species. Small clusters of the same species are shown as different coloured dots on the surface. Notice, for example, how the blue dots only appear on the highest peaks whereas the dark green dots favour the lower altitudes.

The model in not intended to be an accurate biological distribution model, however it is sufficient to generate visually plausible results.

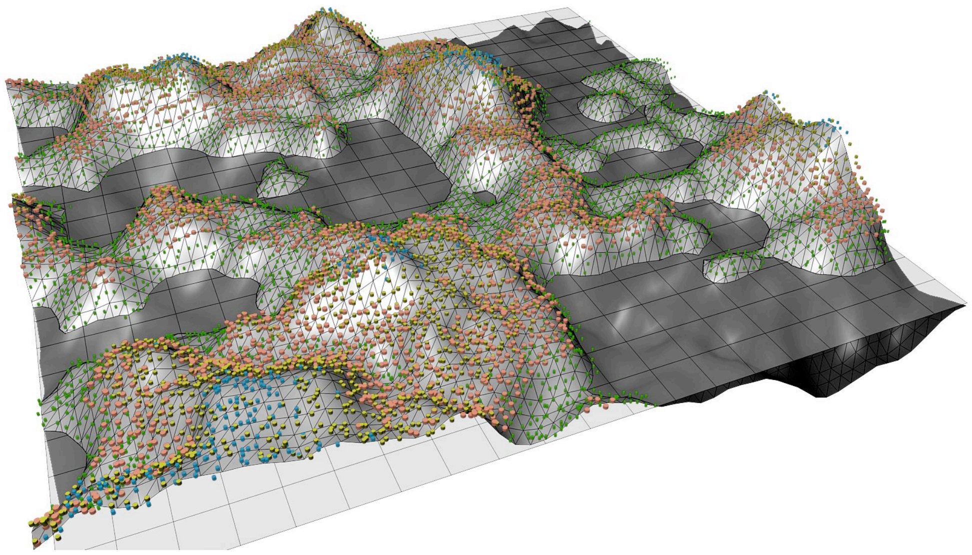


Figure 7-21: Species distribution over the landscape. Clusters of different species are shown as coloured dots for preview purposes. Different colours represent different species, activated as separate L-systems when plants are generated.

7.3.5 Results

Figure 7-22 shows two rendered images of the generic grass model described in Section 7.3.1. Two different variations are used, based on the productions defined in (7.26). One model triggers the production of flowers, the other, seed husks at different stages of development. The mixing of these two models is controlled by the `noise` function. The distribution of individual plant elements follows a Poisson distribution, ensuring minimum spacing between individual plants. For each image all the geometry is created by application of the L-system rules — no external geometry or distribution software is required. The construction of stalks, leaves, husks, florets, etc. is all achieved using generalised cylinder techniques.

Using a similar method to that of introducing variance in stalk growth (Section 7.3.1), other parameters are also varied using specific distributions (uniform, Poisson or normal), these include: flower size, leaf angle, twist, and husk dimensions. In some cases, variance has causal relationships based on simple observation (e.g. wide stems generate bigger flowers). The resultant models do not mimic a set species, the grasses have a similar visual appearance to a number of Australian species, e.g. strap grass, Dianella, or some of the small lilies (*Thysannotus* sp.), however the flowers are reminiscent of the Geraldton Wax. This is based on aesthetic choices — the intent is not botanical realism. However, there is no reason that accurate botanical models could be created by this technique if that was desired.

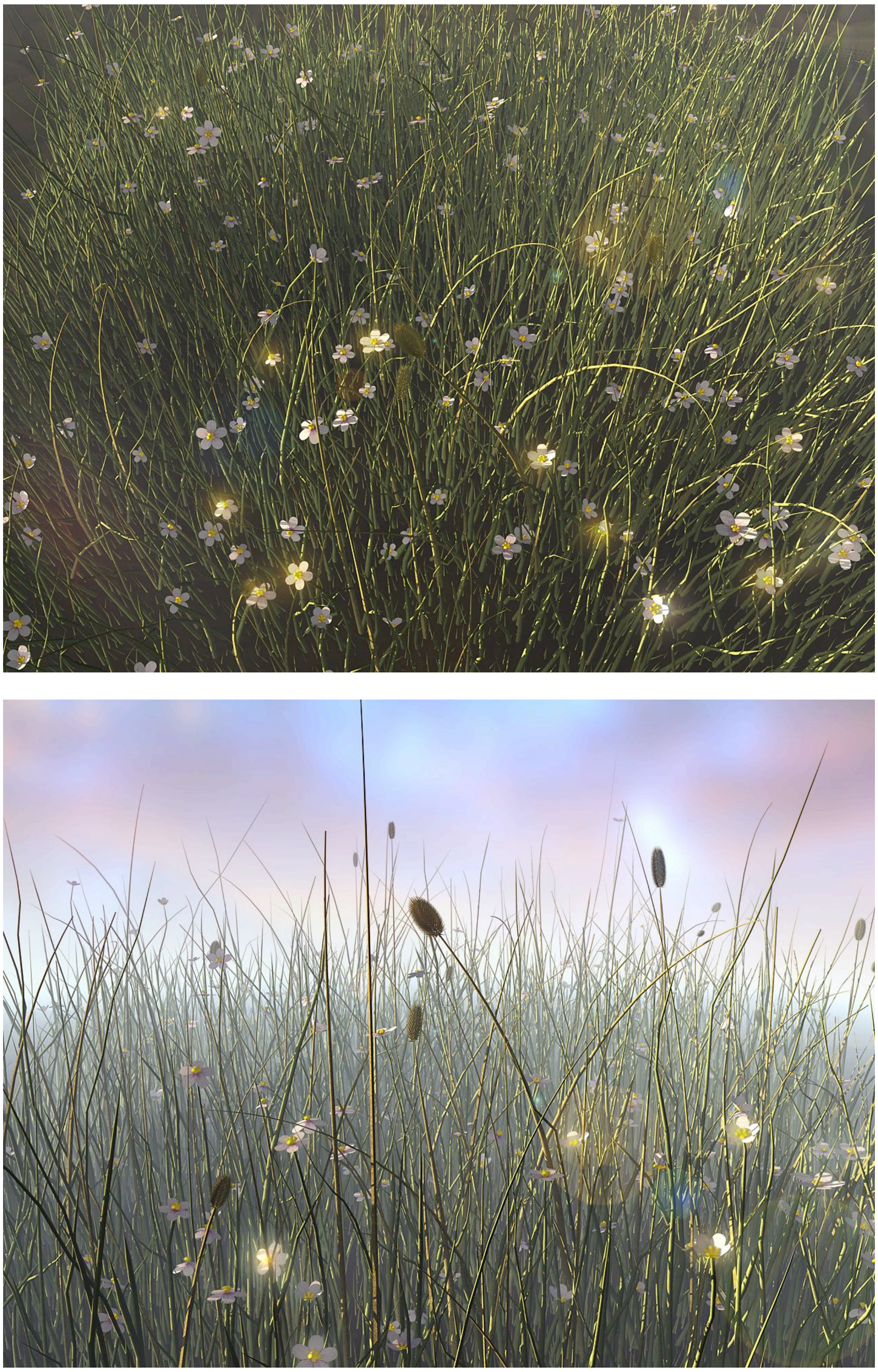


Figure 7-22: *Grass Field.* Two views of the grass model with stochastic distributions.

Figure 7-24 shows some sample images generated using the terrain model with distribution as outlined in Section 7.3.4. As a comparison, the terrain without vegetation is illustrated in Figure 7-23. As can be seen, the vegetation contributes significantly to the complexity of the scene. Render time for each image was approximately 24 minutes on a dual 500Mhz PowerPC computer. There are over 100,000 individual plants in the final model, with approximately 12.2 million polygons. The L-system productions and axioms that specify the scene require less than 4,000 bytes of storage, but generate in excess of 1.56Gb of geometric data — an expansion factor of 390,000:1. The use of the term *database amplification* seems particularly apt in this situation.

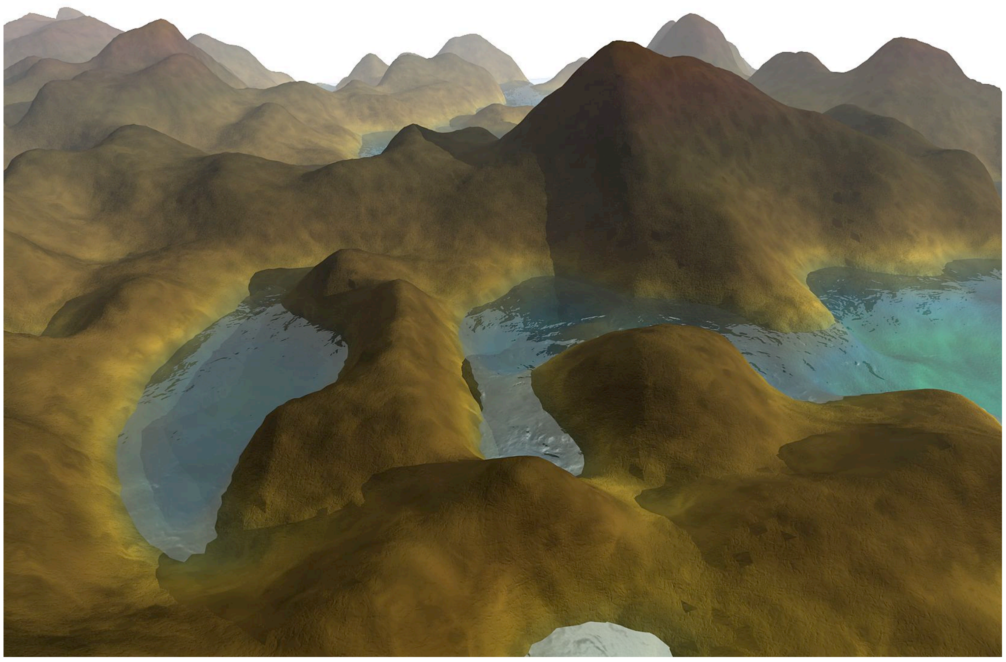


Figure 7-23: Terrain model rendered without vegetation (cf. Figure 7-24). Procedural textures are used to shade the terrain and water. For the terrain, various maps based on the height field provide variation of colour, texture and shading based on elevation.

The terrain model uses a relatively simple heterogeneous model based on the `noise` and `fractalSum` functions. Terrains that are more complex could have been implemented, such as those described in (Musgrave 1994), however given that much of the terrain is covered in vegetation additional terrain complexity would largely go unnoticed, as little of the actual terrain is visible. The given combination of terrain and vegetation is dependent on the specific requirements of the artist or user. What has been demonstrated in this section is that such landscape modelling is possible using L-systems and the functions described in this thesis.

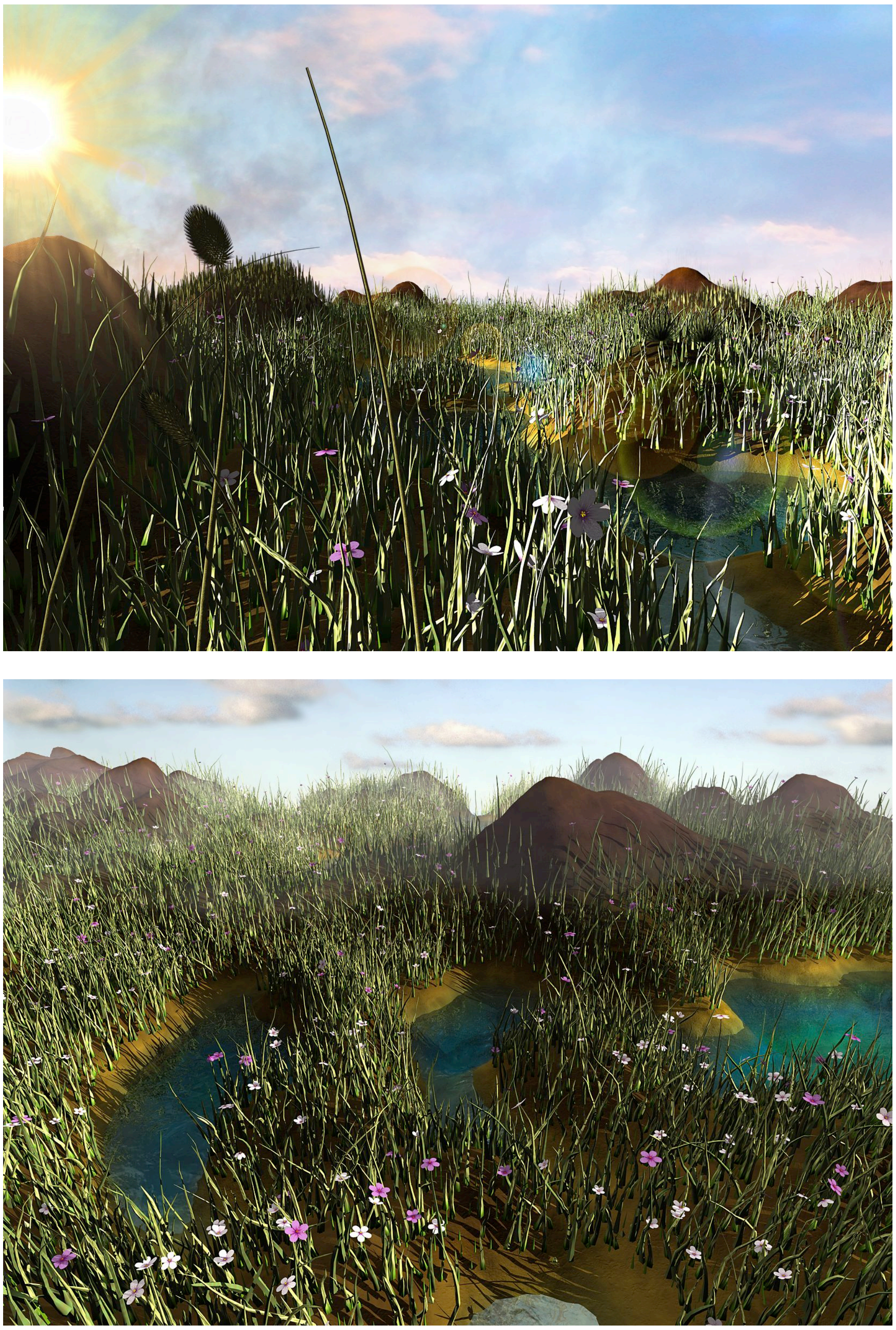


Figure 7-24: Terrain and plants generated using L-systems as described in this section.

8 Animation

Time is that by virtue of which everything becomes nothingness in our hands and loses all real value.

—*Arthur Schopenhauer*

The previous chapter covered the use of L-systems as a generative modelling system suited to three–dimensional form, particularly branching structures and botanical models. This chapter extends the diversity and flexibility of the generative modelling approach by adding functionality in the animation of continuous development.

8.1 Animation using L-systems

The L-system formalisms described in the previous chapters (deterministic, stochastic, context sensitive and parametric) have not considered the continuity of spatial and temporal development. The formalisms on which they are based are discrete in both space and time. That is, productions replace strings simultaneously and at discrete steps. For the modelling of plants and branching structures for biological applications, this may be acceptable. However, in applying L-systems to broader domains, continuous development provides a number of advantages; these will be discussed shortly.

Initially, it appears that visual applications such as animation are in many senses discrete. Film and video animation is a “trick of the eye”, where the rapid succession of static individual images or *frames* gives the illusion of movement. In many forms of music, notes are discrete events chosen from a fixed set of pitch values. Discrete models can be applied to these applications, however as Prusinkiewicz and Lindenmayer (Prusinkiewicz & Lindenmayer

1990, page 133-134) point out, in the case of L-systems, this has a number of disadvantages:

- Models can be constructed using longer or shorter time intervals, but this then becomes part of the model and becomes difficult to modify. Film and video have different sampling rates and interactive graphics display can have not have a guaranteed display (refresh) rate.
- The smooth animation and generation of shapes is easier to specify in the continuous time domain as opposed to adding more complexity to productions to deal specifically with smooth growth and development.
- It is conceptually elegant to separate model development (which is continuous) from model observation (which is discrete).

The symbolic and discrete nature of an L-system alphabet is inherently suited to a stepwise topological and structural description, whereas the developmental aspects may be more suited to a continuous model. For a flexible organic modelling system, continuous components cannot be ignored. In the creative arts, time-based, developmental, real-time interactive systems have become increasingly prominent (Paul 2003). As discussed in Section 3.6, the idea of an artwork as temporal system has a complex and acknowledged history in contemporary art.

The developmental process of *timed L-systems* (defined in the next section) reflects periods of continuous model change punctuated by the instantaneous changes of module divisions or replacements. Prusinkiewicz and Lindenmayer see an analogy with the theory of morphogenesis advanced by Thom (Thom 1975) where development is seen as a piecewise continuous process, punctuated by *catastrophes*.

8.1.1 Related Work

Noser, Thalmann and Turner combined timed L-systems with “vector force fields” to create an animation system that enables simulation of environmental interaction (Noser & Thalmann 1993; Noser, Thalmann & Turner 1992). Their main emphasis was on this environmental interaction rather than as a general animation system. The *RTEvol* system of Nguyen is a procedural modelling system that supports deterministic, stochastic, parametric and timed L-systems with output to popular ray-tracing packages (Nguyen 1997).

Work by Prusinkiewicz, Hammel and Mjolsness introduced *differential L-systems* where module development is controlled by simulating diffusing chemicals within a module that trigger rewriting when they reach a certain concentration (Prusinkiewicz, Hammel & Mjolsness 1993). Elements of this methodology are integrated into the cellular developmental system described in the next chapter.

8.2 Timed, Parametric 0L-systems

Timed L-systems were proposed by Prusinkiewicz and Lindenmayer (Prusinkiewicz & Lindenmayer 1990, Chapter 6) as an extension for achieving continuous development with D0L-systems. The definition here varies slightly in a number of respects:

- Parametric L-systems are the basis for developing the timed extension;
- The birth time parameter may be an expression;
- A "growth" function, here termed the *development function* is used instead of the growth function described in (Prusinkiewicz & Lindenmayer 1990, page 140). This function is used to achieve a more flexible animation system and is further discussed in Section 8.2.1;
- The original formulation was primarily for modelling cellular developmental processes, whereas here it is used for the more general purpose of producing generative animation sequences with the L-system models previously discussed.

Here, parametric 0L-systems (defined in Section 5.3.5.1) are modified to include continuously variable module *ages* that permit continuous temporal and spatial development that is difficult or impossible to achieve with conventional 0L- or nL-systems.

In this case, *modules* consist of *timed symbols* with associated *parameters*. For each module, the symbol also carries with it an *age* — a continuous, real variable, representing the amount of time the module has been active in the derivation string. Strings of modules form *timed, parametric words*, which can be interpreted to represent modelled structures. As with parametric L-systems, it is important to differentiate between *formal modules* used in production specification, and *actual modules* that contain real-valued parameters and a real-valued age. We assume the definitions of formal and actual parameters, logical and arithmetic expressions described in Section 5.3.5 for parametric 0L-systems.

Let V be an alphabet, $\Re$ the set real numbers and $\Re_+$ the set of positive real numbers, including 0. The triple $(s, \lambda, \tau) \in V \times \Re^* \times \Re_+$ is referred to as a *timed parametric module* (hereafter shortened to *module*). It consists of the symbol, $s \in V$, its associated parameters, $\lambda = a_1, a_2, \ldots, a_n \in \Re$ and the *age* of s, $\tau \in \Re_+$. A sequence of modules, $x = (s_1, \lambda_1, \tau_1) \cdots (s_n, \lambda_n, \tau_n) \in (V \times \Re^* \times \Re_+)^*$ is called a *timed, parametric word*. A module with symbol $S \in V$, parameters $a_1, a_2, \ldots, a_n \in \Re$ and age τ is denoted by $(S(a_1, a_2, \ldots a_n), \tau)$.

A *timed, parametric 0L-system (tp0L-system)* is an ordered quadruplet $G = \langle V, \Sigma, \omega, P \rangle$ where:

- V is the non-empty set of symbols called the alphabet of the L-system;
- Σ is the *set of formal parameters*;
- $\omega \in (V \times \Re^* \times \Re_+)^+$ is a nonempty timed, parametric word over V, called the *axiom*, and
- $P \subset (V \times \Sigma^* \times \Re_+) \times C(\Sigma) \times (V \times \mathcal{E}(\Sigma)^* \times \mathcal{E}(\Sigma))^*$ is a finite *set of productions*.

A production $(\underline{a}, C, \underline{\chi})$ is denoted $\underline{a} : C \rightarrow \underline{\chi}$, where the formal module $\underline{a} \in V \times \Sigma^* \times \Re_+$ is the *predecessor*, the logical expression $C \in C(\Sigma)$ is the *condition*, and the formal timed parametric word $\underline{\chi} \in (V \times \mathcal{E}(\Sigma)^* \times \mathcal{E}(\Sigma))^*$ is called the *successor*. Let $(s, \underline{\lambda}, \beta)$ be a predecessor module in a production $p_i \in P$ and $(s_1, \underline{\lambda}_1, \underline{\alpha}_1) \cdots (s_n, \underline{\lambda}_n, \underline{\alpha}_n)$ the successor word of the same production. The parameter $\beta \in \Re_+$ of the predecessor module represents the *terminal age* of s. The expressions, $\underline{\alpha}_i \in \mathcal{E}(\Sigma), i = 1..n$ sets the initial or *birth age*. Birth age expressions are evaluated when the module is created in the derivation string. This nomenclature is illustrated in Figure 8-1.

As with parametric 0L-systems, if the condition is empty the production can be written $\underline{s} \rightarrow \underline{\chi}$. Formal and actual parameter counts must be the same for any given symbol.

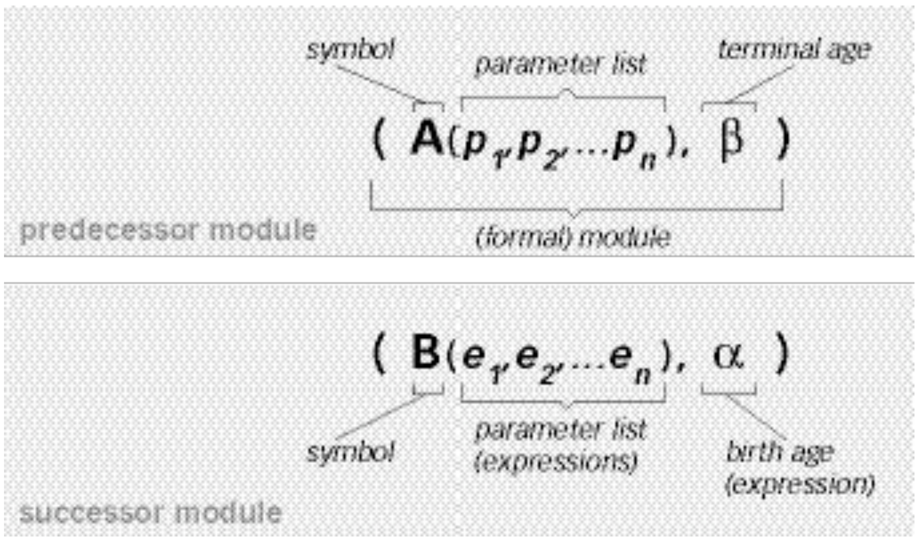


Figure 8-1: Nomenclature for predecessor and successor modules in a timed L-system.

Here are some example productions:

$$\left(A(j,k),3.0\right): j<k \rightarrow \left(B(j*k),0.0\right)\left(C(j+1,k-1),0.5\right) \tag{8.1}$$

$$\left(A(t),3.0\right) \rightarrow \left(A(t+1),3.0/t\right) \tag{8.2}$$

It is assumed:

- For each symbol $s \in V$ there exists at most one value of $\beta \in \Re_+$ for any production $p_i \in P$ where $\left(s,\underline{\lambda},\beta\right)$ is the predecessor in p_i. If s does not appear in any production predecessor then the terminal age of s, $\beta_s = \infty$ is used (effectively the module never dies).
- If $\left(s,\underline{\lambda},\beta\right)$ is a production predecessor in p_i and $\left(s,\underline{\lambda}_i,\underline{\alpha}_i\right)$ any module that appears in a successor word of P for s, then $\beta > \alpha_i$ when $\underline{\alpha_i}$ is evaluated and its value bound to α_i (i.e. the lifetime of the module, $\beta - \alpha_i > 0$).

Development proceeds according to some global time, t, common to the entire word under consideration. Local times are maintained by each module's age variable, τ (providing the relative age of the module). Let $\left(\left(s,\underline{\lambda},\beta\right),C,\left(b_1,\underline{\lambda}_1,\underline{\alpha}_1\right)\ldots\left(b_n,\underline{\lambda}_n,\underline{\alpha}_n\right)\right)$ be a production in *P*. To obtain the derivation string at some global time, t, a *derivation function*, $\mathcal{D}:\left(\left(V\times\Re^*\times\Re_+\right)^+\times\Re_+\right)\rightarrow\left(V\times\Re^*\times\Re_+\right)^*$ is defined:

1. $\mathcal{D}\left(\left(\left(s_1,\lambda_1,\tau_1\right)\ldots\left(s_n,\lambda_n,\tau_n\right)\right),t\right)=\mathcal{D}\left(\left(s_1,\lambda_1,\tau_1\right),t\right)\ldots\mathcal{D}\left(\left(s_n,\lambda_n,\tau_n\right),t\right)$

2. $\mathcal{D}\big((s,\lambda,\tau),t\big) = (s,\lambda,\tau+t)$, if $\tau + t \le \beta$

3. $\mathcal{D}\big((s,\lambda,\tau),t\big) = \mathcal{D}\Big(\big((b_1,\lambda_1,\alpha_1)\ldots(b_n,\lambda_n,\alpha_n)\big),t-(\beta-\tau)\Big)$, if $\tau + t > \beta$ and $C(\underline{\lambda}) \ne 0$

The evaluation of conditions is the same as for parametric L-systems (described in Section 5.3.5). Parameters are independent of module age (i.e. they do not change as the module ages). For *tOL-systems* (Prusinkiewicz & Lindenmayer 1990, page 136) show that this derivation and associated conditions guarantees a derivation for any value of *t*. This scheme also ensures that observation times are independent of the model development itself.

As an example, consider the following timed parametric L-system:

$$
\begin{aligned}
\omega &: \ (A(1),0) \\
p_1 &: \ (A(i),1) : i > 0 : \rightarrow (A(-i),0)(B(i),0) \\
p_2 &: \ (A(i),1) : i \le 0 : \rightarrow (B(i),0)(A(-i),0) \\
p_3 &: \ (B(i),1) \rightarrow (A(i),0)
\end{aligned}
\tag{8.3}
$$

The table below shows how the derivation function is used to evaluate the derivation string at different global times.

Global Time	Derivation Function	Derivation String
$t = 0.5$	$\mathcal{D}((A,1,0),0.5) = (A,1,0.5)$ [by A2]	$(A(1),0.5)$
$t = 1.5$	$\mathcal{D}((A,1,0),1.5) = \mathcal{D}(((A,-1,0.0)(B,1,0)),0.5)$ [by A3]	
	$= \mathcal{D}((A,-1,0.0),0.5)\,\mathcal{D}((B,1,0.0),0.5)$ [by A1]	
	$= (A,-1,0.5)(B,1,0.5)$ [by A2]	$(A(-1),0.5)(B(1),0.5)$
$t = 2.5$	$\mathcal{D}((A,1,0),2.5) = \mathcal{D}(((A,-1,0.0)(B,1,0)),1.5)$ [by A3]	
	$= \mathcal{D}((A,-1,0.0),1.5)\,\mathcal{D}((B,1,0.0),1.5)$ [by A1]	
	$= \mathcal{D}((B,-1,0.0)(A,1,0.0),0.5)$	
	$\mathcal{D}((A,1,0.0),0.5)$ [by A3]	
	$= \mathcal{D}((B,-1,0.0),0.5)\,\mathcal{D}((A,1,0.0),0.5)$	
	$\mathcal{D}((A,1,0.0),0.5)$ [by A1]	
	$= (B,-1,0.5)(A,1,0.5)(A,1,0.5)$ [by A2]	$(B(-1),0.5)(A(1),0.5)(A(1),0.5)$

Table 8–1: Use of the derivation function for the sample L-system.

8.2.1 Development Functions

The words produced by application of productions can undergo a turtle interpretation as described for parametric L-systems in Chapter 6. What remains is to specify a relationship between a module's age, parameters and its interpretation by the turtle system.

Let $m = (s, \lambda, \tau) \in V \times \Re \times \Re_+$ be an actual module composed of the symbol, s, its actual parameters, λ and its current relative age, τ. A timed symbol $s \in V$ may optionally have associated with it a *development function,* $g_s : (V \times \Re^* \times \Re_+) \to \Re$. This function may involve any of the parameters, current age, τ, and the terminal age β of s (determined by the predecessor of the production acting on s). Thus g_s is a real valued function that can be composed of any arithmetic expression $E(\underline{\lambda_s}, \underline{\tau_s}, \underline{\beta_s})$. In addition to the formal parameters supplied, expressions can include the operators, numeric constants, and functions defined in Section 5.3.5. The development function returns a real value, which is then used as a scaling factor for the actual parameter vector λ. That is:

$$\lambda' = g_s \cdot [\lambda] \tag{8.4}$$

The development function is evaluated whenever a module requires turtle interpretation, with parameter vector λ' sent to the turtle, rather than λ as was the case with parametric L-systems. No constraints are placed on the range or continuity of g_s, however if continuity is required when a production is applied (such as the accurate modelling of cellular development), g_s must be monotonic and continuous. These constraints extend to the development functions for those symbols that are related as being part of the successor definition of s.

The development function is specified in the form:

$$g_s(\texttt{parameter\_list}) = \texttt{expression} \tag{8.5}$$

Where `parameter_list` is drawn from the parameters, age and terminal age of s $\{\underline{\lambda_s}, \underline{\tau_s}, \underline{\beta_s}\}$ and `expression` is any valid arithmetic expression as discussed in relation to parametric L-systems (Section 5.3.5).

8.2.1.1 Development Functions and Growth Functions

It is common in the literature to talk of *growth functions* in association with L-system derivation. Growth functions relate the size of the derivation string (number of symbols) with the derivation step n of the string. The growth function, $f_G(n)$ for a D0L-system $G = \langle V, \omega, P \rangle$ has been shown (Rozenberg & Salomaa 1980, pages 33-38) to be:

$$f_G(n) = \sum_{i=1}^{s} P_i(n)\rho_i^n \quad \text{for } n \geq n_0 \tag{8.6}$$

where $P_i(n)$ is an polynomial with integer coefficients, ρ_i a nonnegative integer, and $n_0 = |V|$. Studies have shown that many growth processes observed in nature cannot be described by equation (8.6).

In the development of timed D0L-systems (tD0L-systems), Prusinkiewicz and Lindenmayer require growth functions to relate a module's age with the shape of the entity it represents. Their growth function:

$$f_G(a, \tau) \tag{8.7}$$

Specifies the length of a cell, a as a function of the module's age τ.

They impose continuity requirements to ensure smooth animation of cellular development and continuity of cell sizes in the case of using tD0L-systems to simulate the growth of non-branching filaments. This continuity extends to higher order derivatives in order to eliminate discontinuities. They site the studies of morphogenesis by Thom (Thom 1975) who assumes a differential model a priori as the basis for studying morphogenesis. This theory is furthered in the developmental aspects of differential L-systems (Prusinkiewicz, Hammel & Mjolsness 1993).

The definition of growth functions for D0L-systems as shown in equation (8.6) and that for timed D0L-systems (equation (8.7)) reflects a subtle change in that the original formulation (8.6) was for the ordinal length of a derived string, whereas for timed L-systems it is the relationship between a module's size and its age.

The development function, introduced in Section 8.2.1, is not referred to as a "growth function", even though there are similarities between the development function and the growth function of equation (8.7). This is because the development functions do not literally describe the length of derived strings (or their mapping to cells), rather they provide an association between a module's age, its actual parameters, and the interpretation of that module. In addition, development functions may not necessarily reflect changes in growth, since they permit a wider degree of control under the extended turtle interpretation discussed in Chapter 1.

8.2.2 Growth Examples

Here two examples of the use of timed L-systems and their associated development functions are shown, highlighting their application in modelling animated mechanical and organic structures.

8.2.2.1 A Simple Piston System

This example simulates a simple piston and flywheel mechanism. The piston is connected to the flywheel via an arm of fixed length. The movement of the piston is constrained to the vertical, which drives the wheel in a circular motion (Figure 8-2).

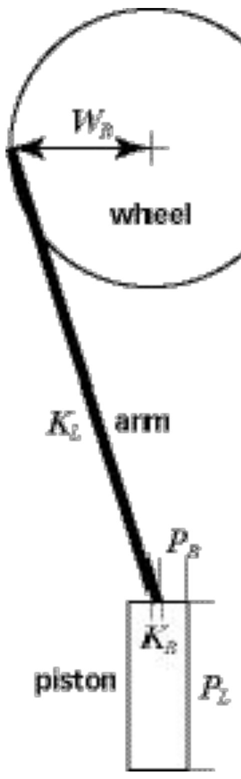


Figure 8-2: Diagram of the piston system.

Figure 8-3 is the L-system used to model the above system. The period of the system is determined by the constant ρ. In-built turtle geometry commands are sufficient to construct all the geometry. Production p_1 builds the system, productions $p_2 - p_5$ cycle the components in loops of period ρ. When a symbol reaches the end of its life, it begins again with age 0 and the cycle continues.

The development functions associated with each timed module provide the information necessary to drive the animation. Thus, the productions provide structure and control, while the development functions control the animation properties of the model.

$$\begin{array}{lll}
\#\mathit{define} & W_R & 50 \\
\#\mathit{define} & K_L & 200 \\
\#\mathit{define} & K_R & 2 \\
\#\mathit{define} & P_R & 10 \\
\#\mathit{define} & P_L & 60 \\
\#\mathit{define} & \rho & 4 \\
\#\mathit{define} & \mathrm{E} & 0.04 \\
\mathit{equiv} & f & \mathit{mov} \\
\mathit{equiv} & + & \mathit{bend} \\
\mathit{equiv} & - & \mathit{turn}
\end{array}$$

$$\omega : \ (\mathit{piston}, 0)$$

$$p_1 : \ (\mathit{piston}, \mathrm{E}) \rightarrow (\mathit{mov}(W_R), 0) \,!(P_R)\, F(P_L) (\mathit{bend}(1), 0) \,!(K_R)\, F(K_L)$$
$$(\mathit{turn}(1), 0)\, f(W_R) + \left(\frac{\pi}{2}\right) [\ \mathit{disc}(0, W_R)\ ]$$

$$p_2 : \ (\mathit{mov}(x), \rho) \rightarrow (\mathit{mov}(x), 0)$$

$$p_3 : \ (\mathit{bend}(x), \rho) \rightarrow (\mathit{bend}(x+1), 0)$$

$$p_4 : \ (\mathit{turn}(x), \rho) \rightarrow (\mathit{turn}(x), 0)$$

$$g_{mov}(\tau_{mov}, \beta_{mov}) = 1 - \cos\left(\frac{2\pi\,\tau_{mov}}{\beta_{mov}}\right)$$

$$g_{bend}(\tau_{bend}, \beta_{bend}) = \sin^{-1}\left(\frac{R \sin\left(\frac{2\pi\,\tau_{bend}}{\beta_{bend}}\right)}{L}\right)$$

$$g_{turn}(\tau_{turn}, \beta_{turn}) = g_{bend}(\tau_{turn}, \beta_{turn}) + \frac{2\pi\,\tau_{turn}}{\beta_{turn}}$$

Figure 8-3: tDOL-system to simulate a simple piston mechanism.

A sequence of animated frames generated by the above L-system is shown below in Figure 8-4. The global time for each frame is shown underneath. While this example is not a true physical dynamics simulation, it does demonstrate how constraints and procedural motion can be specified using the timed formalisms described in this chapter.

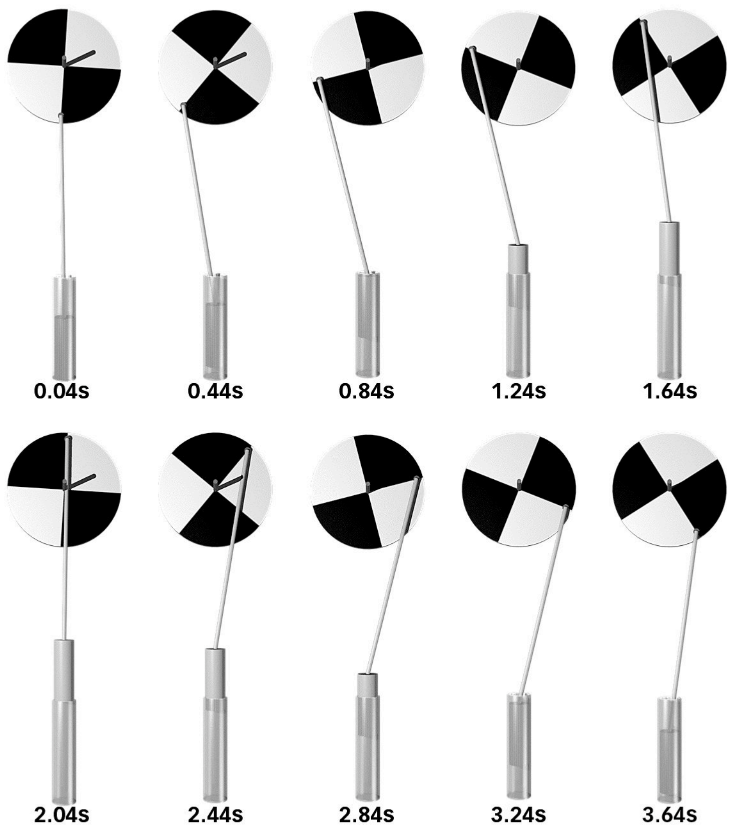


Figure 8-4: Frames from the piston simulation — the geometry and animation is entirely generated from the timed L-system.

8.2.2.2 *Animation of Growth*

I now turn to a more "traditional" example using tpD0L-systems to model the growth in a plant-like object. This example, *Twin-headed Boykinia*, is taken from the interactive animation *Turbulence,* created by the author. In this sequence, a number of evolved and imaginary species of plant-like objects grow on a barren landscape. The sequence required a number of instances of the same "species", and every instance was required to look and animate in a similar fashion, without been identical. This is achieved using timed L-systems and a number of the functions described in Section 6.8, particularly the `noise` and `turbulence` functions.

The full L-system describing the development is quite complex (approximately 35 productions), so in the interests of space and clarity only selected components relevant to the discussion will be detailed here.

There are two main stages in the development of this sequence — *(i)* development of the stem, which after a certain time bifurcates into two segments; and *(ii)* the development of the flower head. The animated development of an individual model is shown below in Figure 8-5.

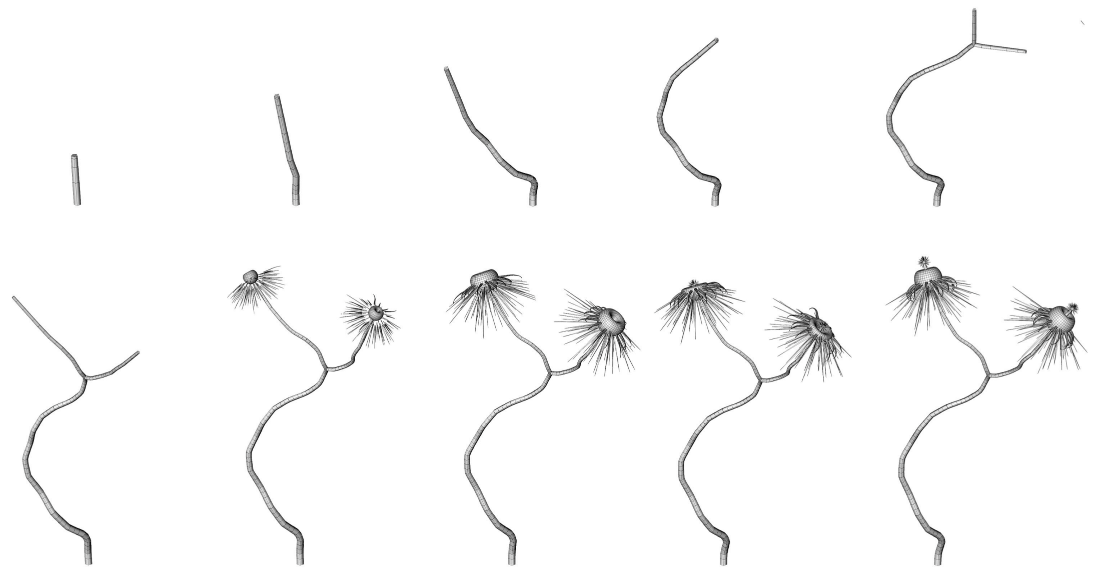


Figure 8-5: Sequence of frames showing the development of the *Twin-Headed Boykinia*. All components that make up the model are developing continuously. Each model is of equal spacing in time (increasing from left to right, top to bottom).

8.2.2.2.1 Stem Growth

The timed L-system describing stem growth is shown below in Figure 8-6. The productions for the module *st* generate the sequence of stem segments and control the branching that occurs after a number of stem segments have developed. The parameters to *st* specify (respectively): plant id (used when generating more than one instance, see Section 8.2.2.2.3), stem segment count, total segment iteration count, segment length, segment radius, and bifurcation level. The constant E specifies the time-period between new segments forming on the stem.

Note the use of random functions that are keyed to particular symbols. In productions $p_1 - p_3$, they control random variation of segment size, orientation and growth direction.

$$
\begin{array}{lll}
\#\,define & \mathrm{E} & 1.0 \\
\#\,define & L_f & 0.97 \\
\#\,define & T_f & 0.96 \\
\#\,define & B_{\max} & 1 \\
equiv & + & br;
\end{array}
$$

$$
\begin{aligned}
\omega &: \left(st\left(n,0,\mathtt{gaussian}_{st}(15,5),\mathtt{gaussian}_{st}(5,0.5),\mathtt{gaussian}_{st}(0.6,0.06)\right),0\right) \\
p_1 &: \left(st\left(n,i,i_{\max},l,t,b\right),\mathrm{E}\right) : i < i_{\max} \rightarrow /\left(\mathtt{random}_{st}(-180,180)\right)!(t)\left(seg\left(l,n-i\right),0\right) \\
&\quad \left(st\left(n,i+1,b,lL_f,tT_f,b\right),0\right) \\
p_2 &: \left(st\left(n,i,i_{\max},l,t,b\right),\mathrm{E}\right) : i \ge i_{\max} \quad \&\& \quad b < B_{\max} \rightarrow [\ \left(br\left(\mathtt{random}_{br}(40,70)\right),0\right) \\
&\quad \left(st\left(n+1,0,b*\mathtt{random}_{st}(0.6,0.9),lL_f,tT_f,b+1\right),0\right)\ ] \\
&\quad [\ \left(br\left(\mathtt{random}_{br}(-40,-70)\right),0\right) \\
&\quad \left(st\left(n+1,0,b*\mathtt{random}_{st}(0.6,0.9),lL_f,tT_f,b+1\right),0\right)\ ] \\
p_3 &: \left(st\left(n,i,i_{\max},l,t,b\right),\mathrm{E}\right) : i \ge i_{\max} \quad \&\& \quad b > B_{\max} \rightarrow /\left(\mathtt{random}_{st}(-180,180)\right)!(t) \\
&\quad \left(fwr(t),0\right)
\end{aligned}
$$

Figure 8-6: Fragment of the timed L-system controlling stem development.

The module *seg* controls the actual construction of stem segment geometry. It uses generalised cylinders that have animated twisting and bending. The development function follows a parameterised exponential growth function.

$$
\begin{array}{lll}
\#\,define & t_{seg} & 0.5 \\
\#\,define & k_s & 4.2 \\
\#\,define & k_n & 0.1 \\
equiv & C & seg \quad seg_b \\
equiv & + & bend \\
equiv & / & twist
\end{array}
$$

$$
p_4 : \left(seg(l),t_{seg}\right) \rightarrow \left(bend(4l),0\right)\left(twist(4l),0\right)\left(seg_b(l),t_{seg}\right)
$$

$$
\begin{aligned}
g_{seg}\left(\tau_{seg},\beta_{seg}\right) &= 1 - e^{-k_s \frac{\tau_{seg}}{\beta_{seg}}} \\
g_{seg_b}\left(\tau_{seg_b},\beta_{seg_b}\right) &= g_{seg}\left(\tau_{seg_b},\beta_{seg_b}\right) + \\
&\quad k_n\left(\mathtt{smoothstep}\left(t_{seg},2t_{seg},\tau_{seg_b}\right)\ \mathtt{noise1D}_{seg_b}\left(\tau_{seg_b}\right)\right)
\end{aligned}
$$

The module *seg* grows until maturity (time t_{seg}), whereupon it is replaced by module seg_b, representing the mature segment.[59] At this stage, the growth function changes to incorporate a noise component. As seg_b has no specified

[59] Even though the module changes, interpretation of *seg* and seg_b are the same. There is no discontinuity because the growth functions form a continuous function.

production, its terminal age is assumed to be infinite (the timed equivalent of the identity production). The growth functions for modules *bend* and *twist* (not shown) use a similar noise based methods to animate the bending and twisting of each segment, both during development and as a mature plant.

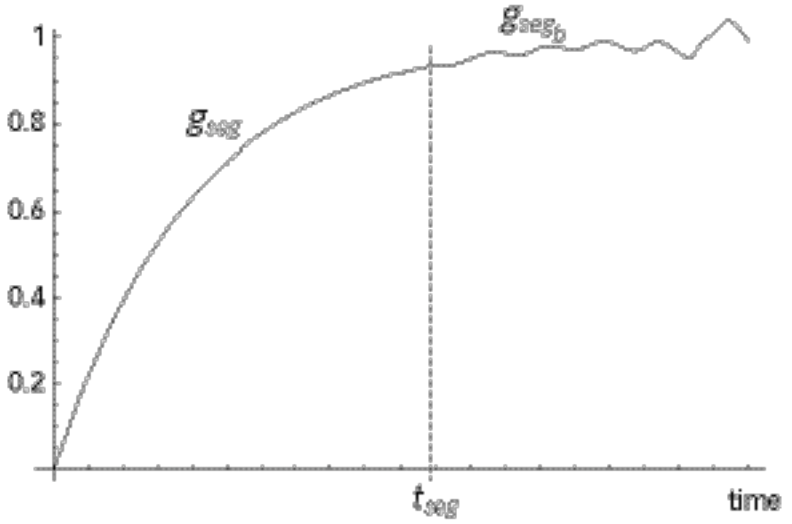


Figure 8-7: Graph of development functions g_{seg} and g_{seg_b}. The functions are chosen to provide continuous growth in animated sequences. The segment reaches maturity at t_{seg}.

8.2.2.2.2 Flower Growth

The module *fwr* initiates growth of the flower head. It consists of a number of animated components. The main head uses a set of three pre-defined surfaces, which are interpolated by growth functions. Generalised cylinders are used to model the thorn and spike components. Note how these features animate in both shape and size.

A Bessel function of the first kind is used to model the scaling and animation of the flower head (Figure 8-8). Bessel functions are often used to solve motion equations for physical systems (Kreyszig 1999, p. 218), and here the use of the function gives the animation a damped-spring-like quality (which the equation represents), as the head “puffs” up rapidly in size and then pulsates in an oscillating rhythm, slowly damping down as the element ages. Visually similar behaviour is observed in time-lapse sequences of real plants, as they respond to the rhythms of the day/night cycle.

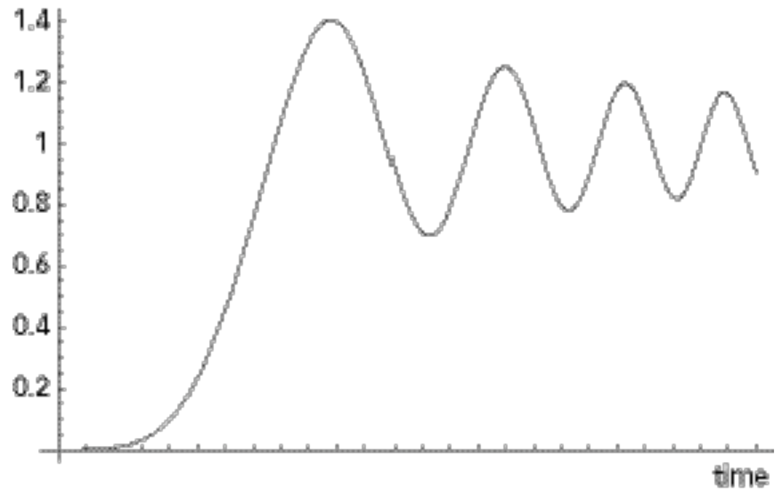


Figure 8-8: Development function used for animation of the flower head based on a Bessel function of the first kind.

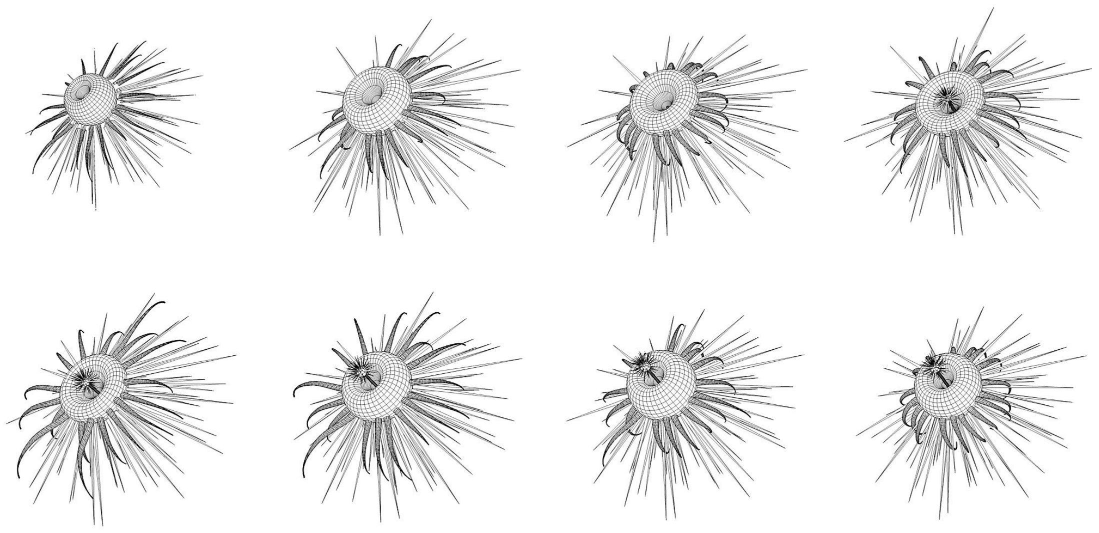


Figure 8-9: Growth of the flower head. The geometry contains both pre-defined surfaces and generalised cylinders.

8.2.2.2.3 Multiple Instances of the Same Species

By keying random number generation functions (Section 6.8.2), variation can be achieved amongst individuals of the same species (i.e. generated by the same L-system). By associating a unique id with each plant random number generation keys change with each instance, thus creating variation amongst the models generated using the same deterministic L-system.

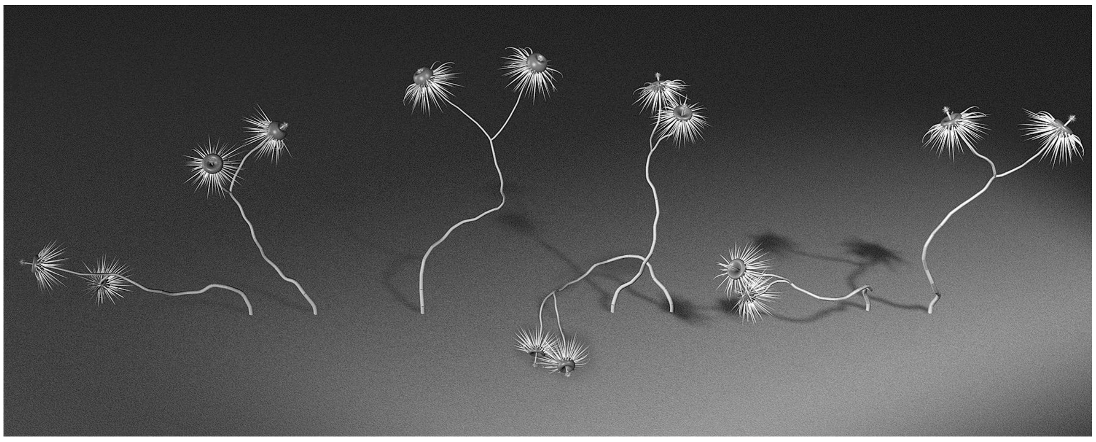


Figure 8-10: Various instances of the *Twin-Headed Boykinia* model showing how variation is achieved using the stochastic functions of the L-system model described in this thesis. Each plant was generated from the same L-system, but with a unique id used to seed the keyed random number generation functions.

Frames from the final animated sequence generated for *Turbulence* are shown below, the L-system animating both geometry and texture information. The sequence shows approximately 2.5 seconds of animation.

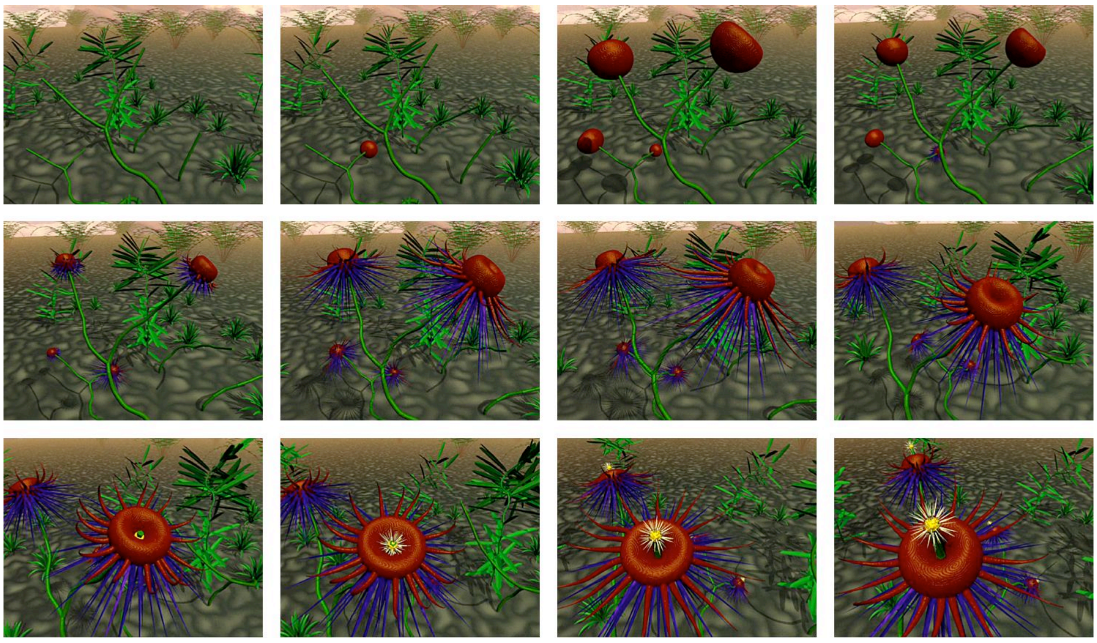


Figure 8-11: Frames showing the development of the *Twin-headed Boykinia* sequence from the interactive animation *Turbulence*.

8.3 Detaching Turtle State

In Section 6.4, the use of turtle command "%" (break symbol), was introduced. The use of this function is now further described in relation to animation and timed L-systems.

In some animation sequences, it is necessary to break the inherited dependence of one geometric structure from another, where previously that dependence was required. For example, a seed that grows inside a seed-case moves with that case, until the time that the case opens and the seed falls out. From this point in time, its movements are independent of its former parent. Animating such events using timed L-systems can be difficult, because symbols are interpreted relative to the current turtle state, not at absolute positions or orientations.

The break turtle command ("%") facilitates the independent movement of objects where previously there was dependence. The command works by triggering a turtle interpretation of all modules in the current module list up until the specific "%" module's position. This event occurs at the module's birth and happens only once in it's lifetime. This non-standard feature requires specific testing during the development of modules (see the algorithm in Figure 8-28).

An example of the use of this break command is shown in the timed L-system of Figure 8-13. This is a simplified section of a more complex animated sequence. In the interest of clarity, a number of productions not essential to this discussion have been removed. Figure 8-14 shows frames from the animated sequence generated by the complete version of this L-system.

The sequence consists of moving branching structures, which grow and form "seed-case" structures at the end (Figure 8-12 shows these features). The decision to form these cases is based on a stochastic rule (not shown). The cases grow and after a certain time they mature and are able to fire seeds at intervals.[60] Once a seed has been fired, it is no longer geometrically parented to the case, hence the need for the break module.

[60] Although this sequence is not based on any real plant behaviour, in a number of species, seed dispersal is achieved by pods exploding and seeds being shot out. For example, Leafy Spurge (*Euphorbia esula*) can project seeds at distances of up to 4.6 meters!

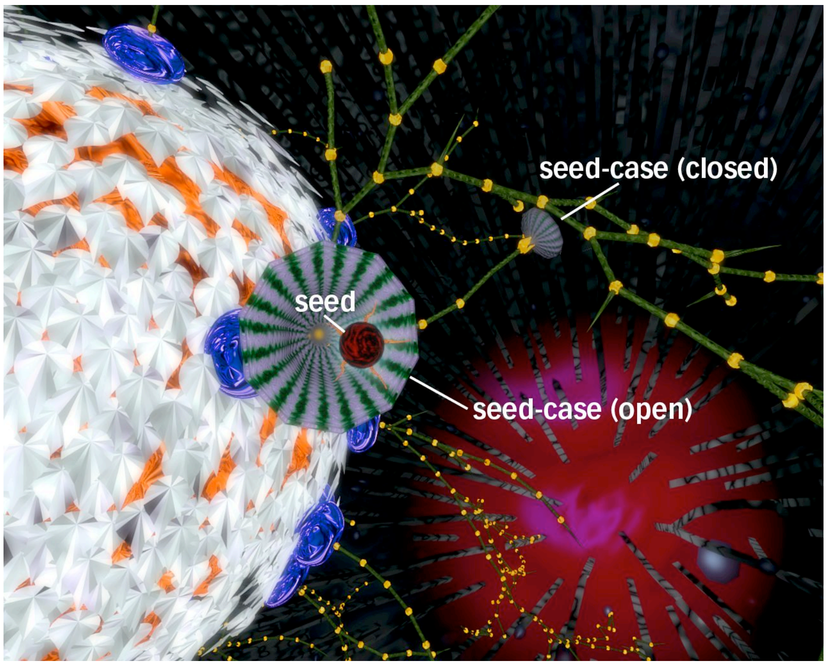


Figure 8-12: Still image of the 'spitter' sequence from the interactive animation *Turbulence* showing seeds and seed-cases referred to in this section.

The seed-cases themselves are modelled using generalised cylinders and undergo a complex animated sequence of "spitting" out a seed. The productions shown in Figure 8-13 begin at this stage essentially represent the control mechanism for this "spitting" sequence (animation of the case is not included here). Productions $p_2 - p_4$ control the growth of the mature seed-case stem. Productions $p_5 - p_7$ control the firing of seeds at random intervals, the module *eject* triggering the ejection of the seed and a spray of "dust" as the seed is ejected.

Production p_{10} shows how the "%" module is used. First, the turtle state is pushed (note the use of the `equiv` command for this purpose, giving the *push* module its own symbol name but equivalent function to the "[" symbol). The turtle dependency broken and two key modules, *Explode* and *seed* instantiated to begin the development of the particle "dust" and seed respectively. These two components are ejected from the seed-case and move independently of their former parent (the actual productions that create the particle dust explosion and seed are not included in the L-system of Figure 8-13). Productions $p_{11} - p_{15}$ terminate the symbols at the end of their development (they are replaced by the null symbol (ε), i.e. removed). It is essential all these modules be removed at the same time to maintain coherence in the animated sequence.

$\text{equiv [} \mathit{push};$

$\text{equiv] } \mathit{pop};$

$\text{equiv F } j\text{Grow } j\ \mathit{jDelay};$

$\text{equiv } \mathit{spit}\ \ \mathit{spitDelay};$

$\#\mathit{define}\quad S\_DLY\quad 1.3$

$\#\mathit{define}\quad S\_TIME\quad 1.5$

$\#\mathit{define}\quad R\quad 4.0$

$\#\mathit{define}\quad QUANT\quad 0.02$

$\#\mathit{define}\quad E\_DLY\quad 0.855$

$\#\mathit{define}\quad E\_TIME\quad 6.0$

$\#\mathit{define}\quad N\quad 8$

$$\omega:\quad \left(\mathit{spitSeq}(n),0\right)$$

$$p_1:\quad \left(\mathit{spitSeq}(n),S\_DLY\right): n \le 4 :\rightarrow \left(F(n),0\right)\left(\mathit{sphere}(1),0\right)$$

$$p_2:\quad \left(\mathit{spitSeq}(n),S\_DLY\right): n > 4 :\rightarrow \left(F(n),0\right)\left(\mathit{jGrow}(R),0\right)\left(\mathit{sphere}(1),0\right)$$
$$\left(\mathit{diskSeq},0\right)\,[\,\left(\mathit{spitGrow}(G),0\right)\left(\mathit{spit}(1,n),0\right)\,]$$

$$p_3:\quad \left(\mathit{jGrow}(l),S\_TIME\right)\rightarrow\left(\mathit{jDelay}(l),0\right)$$

$$p_4:\quad \left(j(l),S\_TIME\right)\rightarrow\left(\mathit{jDelay}(l),0\right)$$

$$p_5:\quad \left(\mathit{spit}(s,n),S\_TIME\right)\rightarrow\left(\mathit{spitDelay}(s,n),0\right)$$

$$p_6:\quad \left(\mathit{spitDelay}(s,n),QUANT\right)\xrightarrow{0.03}\left(\mathit{spit}(s,n),0\right)\,[\,\left(\mathit{eject},0\right)\,]$$

$$p_7:\quad \left(\mathit{spitDelay}(s,n),QUANT\right)\xrightarrow{0.97}\left(\mathit{spitDelay}(s,n),0\right)$$

$$p_8:\quad \left(\mathit{jDelay}(l),QUANT\right)\xrightarrow{0.03}\left(j(l),0\right)$$

$$p_9:\quad \left(\mathit{jDelay}(l),QUANT\right)\xrightarrow{0.97}\left(\mathit{jDelay}(l),0\right)$$

$$p_{10}:\quad \left(\mathit{eject},E\_DLY\right)\rightarrow\left(\mathit{push},0\right)\left(\%,0\right)\left(\mathit{Explode}(E\_DIST),0\right)\left(\mathit{seed},0\right)\left(\mathit{pop},0\right)$$

$$p_{11}:\quad \left(\%,E\_TIME\right)\rightarrow\varepsilon$$

$$p_{12}:\quad \left(\mathit{push},E\_TIME\right)\rightarrow\varepsilon$$

$$p_{13}:\quad \left(\mathit{pop},E\_TIME\right)\rightarrow\varepsilon$$

$$p_{14}:\quad \left(\mathit{Explode}(n),E\_TIME\right)\rightarrow\varepsilon$$

$$p_{15}:\quad \left(\mathit{seed},E\_TIME\right)\rightarrow\varepsilon$$

Figure 8-13: Timed L-system fragment for the ‘spitter’ sequence shown in Figure 8-14.

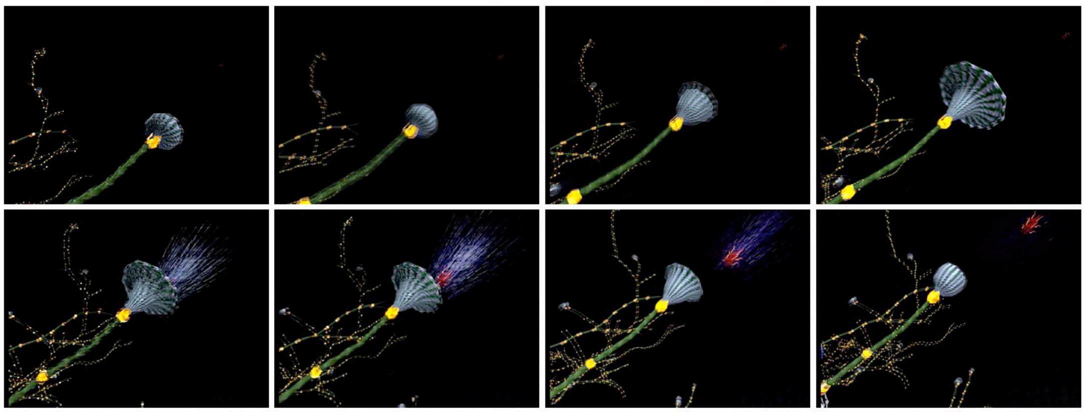


Figure 8-14: 'Spitter sequence' from the interactive animation *Turbulence*. A seed-like object (red) is forcefully emitted from the trumpet-like seed-case structure. The seed then becomes an independent body, with movement in space independent of its former parent. A shower of particles also accompanies the seed as it is emitted.

8.4 Animation of Legged Gaits

The animation of legged gaits has received considerable attention in robotics and graphics research (Badler, Barsky & Zeltzer 1991; Featherstone 1988; Girard & Maciejewski 1985; McKenna & Zeltzer 1990; Raibert & Hodgins 1991). There are many different methods and applications for the generation of legged figure gaits. In this section, I show how the developmental L-system model described in this chapter can be used to model legged gaits. The L-system encodes both body architecture and control specification, allowing an extremely flexible and concise method for generating animated sequences of legged figures.

8.4.1 Techniques for Legged Figure Animation

Apart from general key-framing techniques used in computer animation, basic goal-directed movements of an articulated figure are generally solved using *inverse kinematics*. Using this technique, a set of joint angles are calculated which place the limb at the appropriate position and orientation (the goal), usually calculated by finding the inverse of the Jacobian matrix of joint angles. This calculation can be done incrementally from the base segment (Korein & Badler 1982).

In contrast, *forward kinematics* are used to find the world space coordinates of the end of the limb, given the set of joint angles (Denavit & Hartenberg 1955). The inverse kinematic problem is generally more difficult to solve than the forward kinematic one, since for an arbitrary configuration the system is frequently under-constrained.

Legged gaits may also be driven by *forward dynamics* where the physical dynamics of a linked system is calculated from a set of given input torques. Forward dynamics solvers may be combined with some form of control system that simulates the use of muscles to control an animal's gait. Such a system may also include a general physical dynamics solver, to give physically realistic motion and behaviour. For example, the PODA system of Girard optimises limb movement using both kinematics and dynamics based variables and constraints (Girard 1991). The technique described here creates a control system for limb animation, but the simulation is not fully physical. The primary emphasis is on construction and control of the legged motion using the extensions to L-systems described in this thesis.

8.4.2 Leg and Body Configuration

For the example detailed in this section, we will consider the representation of an "animal" with multiple rigid body segments, each connected with a 2-degree of freedom (DOF) articulation (Figure 8-15).

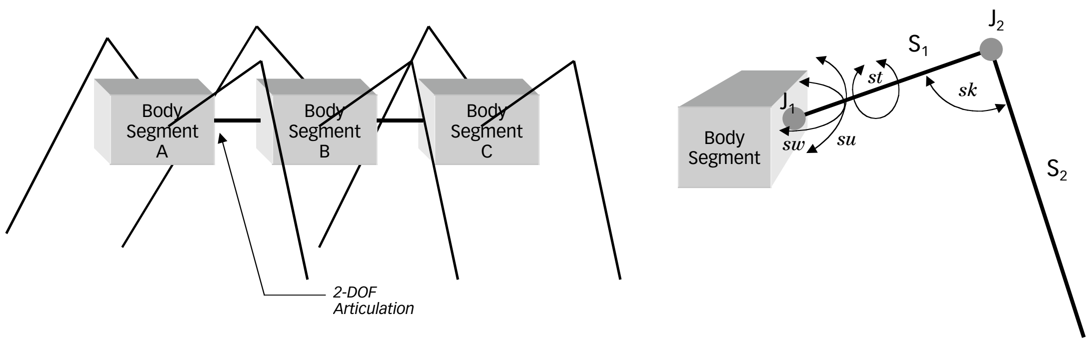


Figure 8-15: Legged animal composed of multiple articulated body segments (left) and detail of an individual segment's joint configuration (right).

Each body segment has two multi-jointed legs, at opposite sides of the body segment. The leg detail is also shown in Figure 8-15. A *leg* is composed of two rigid limb segments, S_1 and S_2. S_1 is attached to the body segment by a 3-DOF joint, J_1. The joint J_2 between S_1 and S_2 has 1-DOF.

The key advantage of an L-system specification is in the flexibility of body and limb design and specification. Similar techniques have been used as a general system to evolve novel designs of articulated figures (Sims 1994a, 1994b). Arbitrary joint, limb, and body segment configurations can be achieved by modifying the productions that generate these elements. The generalised cylinder techniques (discussed in Section 6.6) are used to model the complex limb and body geometries of the creature.

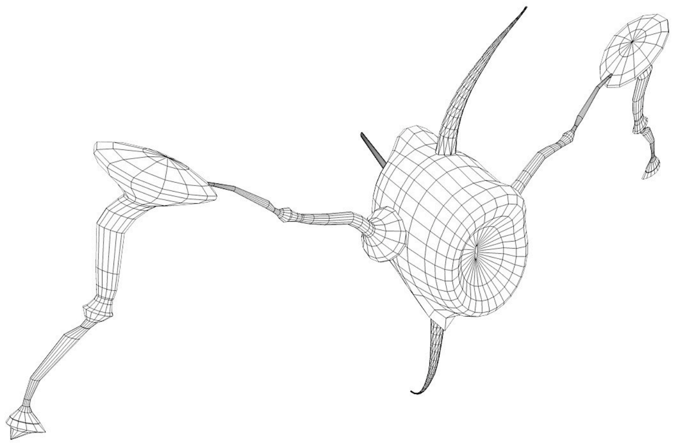


Figure 8-16: Individual body segment showing the geometry created using generalised cylinders. The spikes at the top and bottom of the body segment are also animated as the creature moves.

In the simulation described here, the motor control structure is also specified by the timed L-system, forming a simple finite state machine that drives the movement of the body segments and legs. Constraints on the movement of individual limbs are set within the development functions for each joint. The control system will be the principle focus of the description that follows.

8.4.3 Animation of a Single Leg

To illustrate how this scheme works, I will focus on the construction and animation of a single leg. The L-system shown in Figure 8-17 captures the essential components. The modules *sw*, *su* and *st* represent the motor control of S_1 (3-DOF articulation), while *sk* represents the joint angle between S_1 and S_2. The module *ls* (leg segment) calls a complex set of productions to create the geometry of the leg using generalised cylinders (these productions are not show for the sake of clarity). This module's parameter specifies the overall length of the leg segment and the turtle is placed at the position of the next joint upon completing of the geometric construction of the leg segment. The *ls* module is not timed due to the fixed structure of the leg segment itself. This constraint means that the geometric data can be cached to avoid regeneration across multiple locations and time steps. The constant R_{S12} is the ratio of size between the upper (S_1) and lower (S_2) leg segments.

$$
\begin{array}{lll}
\#define & \beta_{ls} & \varphi \\
\#define & \beta_{sw} & \varphi \\
\#define & \beta_{su} & \varphi \\
\#define & \beta_{st} & \frac{\varphi}{5} \\
\#define & \beta_{sk} & \varphi \\
\#define & R_{S12} & \frac{3}{2} \\
equiv & + & sw \\
equiv & \wedge & su \\
equiv & / & st \\
equiv & \& & sk
\end{array}
$$

$$
p_1 : \left(leg(\phi,l),\beta_{ls}\right) \rightarrow \left(sw(1,\phi),0\right)\left(su(1,\phi),\frac{3\beta_{su}}{4}\right)\left(st(1,\phi),0\right) ls(l)
$$

$$
\left(sk(1),0\right) ls\left(R_{S12}l\right)
$$

$$
p_2 : \left(sw(n,\phi),\beta_{sw}\right) \rightarrow \left(sw(n,\phi),0\right)
$$

$$
p_3 : \left(su(n,\phi),\beta_{su}\right) \rightarrow \left(su(n,\phi),0\right)
$$

$$
p_4 : \left(st(n,\phi),\beta_{st}\right) \rightarrow \left(st2(n,\phi),\beta_{st}\right)
$$

$$
p_5 : \left(sk(n,\phi),\beta_{sk}\right) \rightarrow \left(sk(n,\phi),0\right)
$$

$$
p_6 : \left(st2(n,\phi),3\beta_{st}\right) \rightarrow \left(st(-n,\phi),0\right)
$$

Figure 8-17: L-system to specify a single leg configuration. The cycle time of a single step is specified by the variable φ.

An individual *walk cycle* represents the movement of the leg system over a single gait. The total time for this cycle is represented by the variable φ, with individual controllers using this time or a rational ratio of it to ensure cyclic animation. For example, the twisting motion of the leg (module *st*) is separated into three distinct components which sum to the gait cycle time.

The parameter ϕ represents the phase of the animation cycle. As each body segment is added the phase of the walk cycle is shifted to ensure correct motion relative to the position of the segment. The parameters *n* and *l* control the magnitude of the gait and the leg respectively.

The productions of Figure 8-17 specify a simple state machine that encodes the geometric and temporal structure of the leg gait. The remaining information required is the associated development function for each module, detailed below.

$$
\begin{array}{lll}
\#\mathit{define} & R_{sw} & 0 \\
\#\mathit{define} & A_{sw} & 0.378\pi \\
\#\mathit{define} & R_{su} & 0.056\pi \\
\#\mathit{define} & A_{su} & 0.139\pi \\
\#\mathit{define} & R_{st} & 0 \\
\#\mathit{define} & A_{st} & 0.056\pi \\
\#\mathit{define} & R_{sk} & 0.389\pi \\
\#\mathit{define} & A_{sk} & 0.194\pi
\end{array}
$$

$$g_{sw}\left(\tau_{sw},\beta_{sw},\phi\right)=R_{sw}+A_{sw}\sin\left(\frac{2\pi\tau_{sw}}{\beta_{sw}}+\phi\right)$$

$$g_{su}\left(\tau_{su},\beta_{su},\phi\right)=\mathrm{if}\left(\tau_{su}<\frac{\beta_{su}}{2},0,R_{su}-A_{su}\left(\cos\left(2\pi\left(\frac{\tau_{su}}{\beta_{su}}-0.5\right)+\phi\right)+1\right)\right)$$

$$g_{st}\left(\tau_{st},\beta_{st},\phi,n\right)=R_{st}+\mathrm{sign}(n)A_{st}\sin^2\left(\frac{\pi\tau_{sw}}{\beta_{sw}}+\phi\right)$$

$$g_{sk}\left(\tau_{sk},\beta_{sk},\phi\right)=\mathrm{if}\left(\tau_{sk}<\frac{\beta_{sk}}{2},0,R_{sk}-A_{sk}\left(\cos\left(4\pi\left(\frac{\tau_{sk}}{\beta_{sk}}-0.5\right)+\phi\right)+1\right)\right)$$

Figure 8-18: Key development functions for the gait of a single leg.

These functions approximate the inverse kinematic solution of the system for a walking gait. The use of periodic functions ensures that phase and perfect cycling are easily accommodated, as required for a system that must coordinate a large number of legs in a coherent fashion. In the actual generated sequences, noise perturbations based on the age of the creature are used to introduce variation and a “natural” feel into the walk cycle.

There are no development functions for the *leg* module, since the life cycle of this module does not have any interpretation in the derivation string — it represents the overall gait production for a single leg animation cycle.

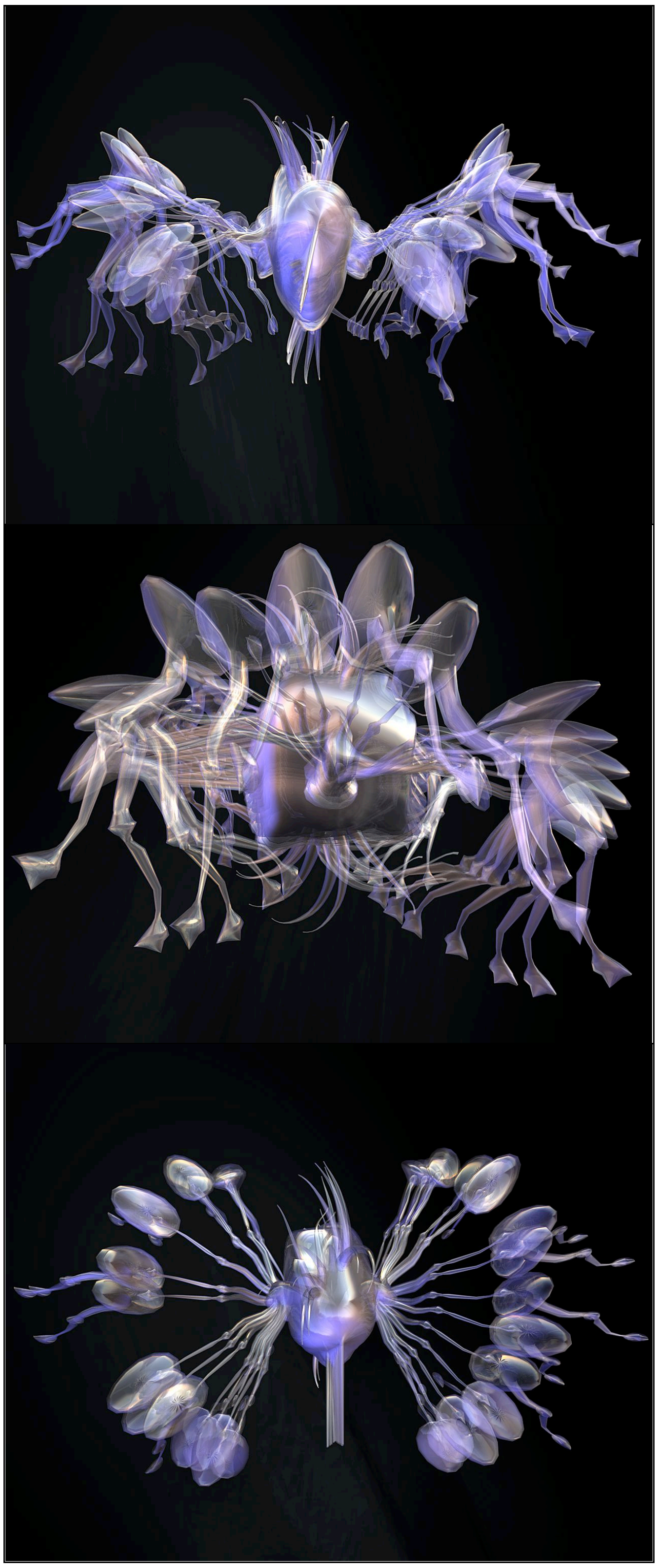


Figure 8-19: Multiple exposure of a single body segment showing the gaits of the legs from the front, side and top.

Sample frames from the animation of a single body segment are shown in Figure 8-19. Individual frames are superimposed over each other to show the range of leg positions over time in a single still image.

8.4.4 Building Multiple Body Segments

The construction of multiple body segments is straightforward. Each segment is built by modules that repeat the productions for an individual body segment, shifting the segment position, in addition to the phase of the body and leg gaits, for each segment. Additional modules and productions control the generation and animation of the head and tail segments. The kinematic animation of connected segments is achieved in a similar manner to that of the individual legs. Figure 8-20 shows a still frame from an animated sequence of a four-segment creature running. Longer or shorter creatures can be synthesised by changing the number of body segments simply by changing a parameter to the module that controls the count of generation of segments.

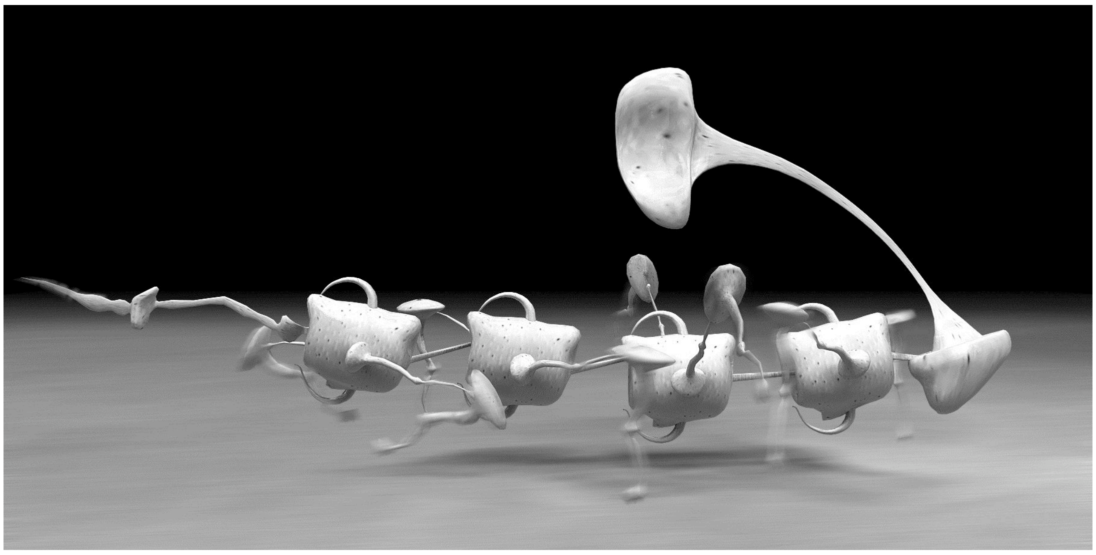


Figure 8-20: Still frame showing the legged creature running. Both the geometry and animation are generated using the timed L-system techniques described in this thesis.

Multiple instances of the individual walking creature can be instantiated using similar techniques to that described in Section 8.2.2.2.3. The figures below show a herd of walking creatures (Figure 8-21) and a sequence of individual frames from the animation *Turbulence* (Figure 8-22). All models were produced using the techniques described in this chapter.

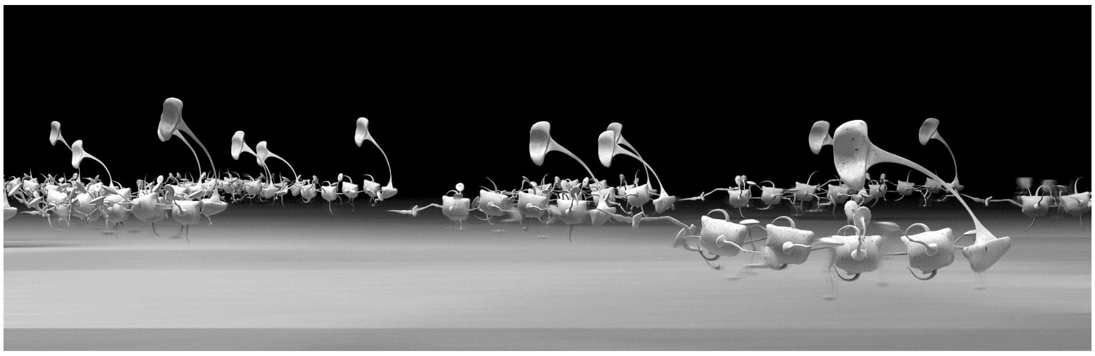


Figure 8-21: A herd of walking creatures created using the timed L-system model described in this chapter.

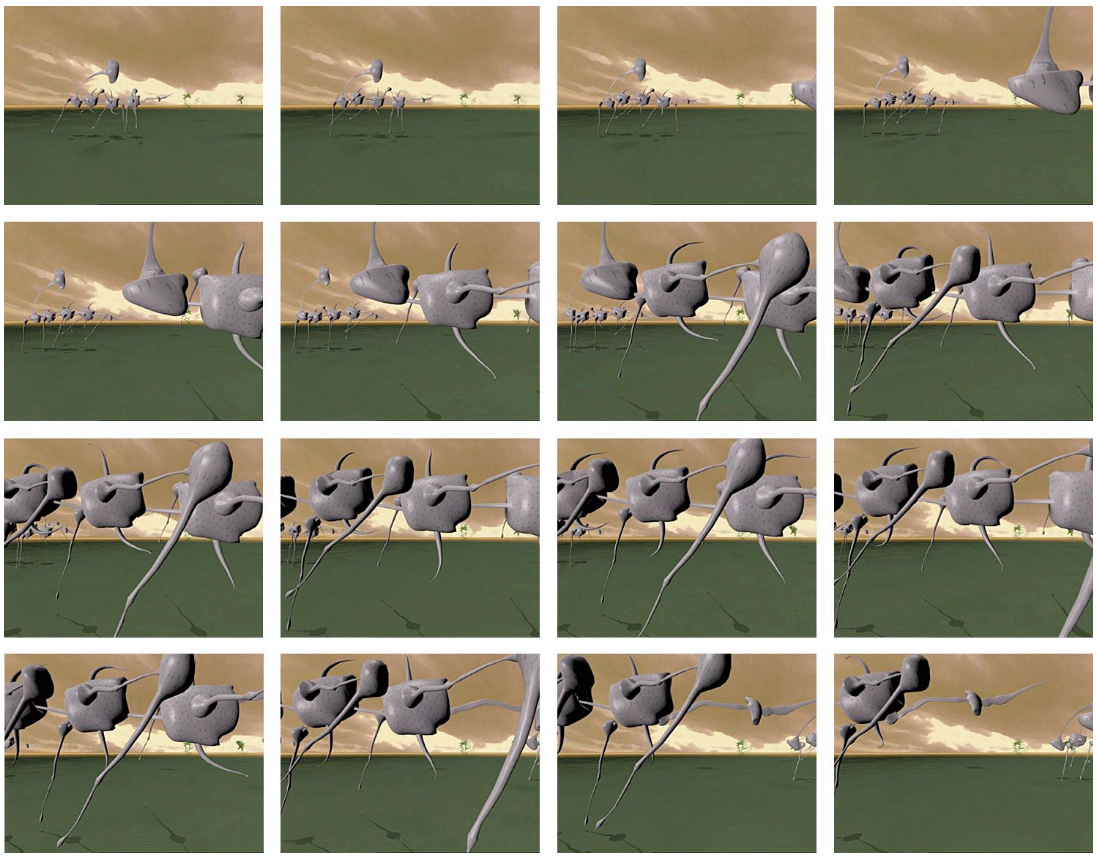


Figure 8-22: Individual frames showing the walking creature from the interactive animation *Turbulence* (see also the colour plates in the appendix).

8.4.5 Discussion

The animated gait system described here has some limitations. It is not as general a system as that designed specifically for dynamic animation of legged figures, since no real physical dynamics are calculated in forming a solution, and the gait phases have been simplified.[61] This is why other systems usually incorporate a physical dynamics solver and only use generative grammars such as L-systems to specify body architecture (Hornby & Pollack 2001a). In addition, the walk cycle illustrated is fixed to operate on flat terrain. By animating the variable φ the gait cycle speed may be adjusted, however extreme adjustments result in unnatural animation, since at high and low speeds the gait of a real animal would change significantly.

Nonetheless, what is significant is that developmental systems can be used for the generative specification of controllers and architectures (mapping to animation and geometry) under a single, unified representation.

The example here also demonstrates how a single "flat" specification can become complex and difficult to modify for the artist or animator, since they must deal with multiple levels of abstraction (animals, bodies, body segments, legs, limbs, etc) as a series of productions. For complex systems involving large numbers of modules, productions, and development functions, this can become a limitation. For example, the body *structure* of the walking creature is conveniently specified in a hierarchy. Complex productions generating geometry such as the legs, require a large number of symbols to be processed in parallel which is why the system described here allows mixing of timed and non-timed modules.

This challenge in modelling complex, biologically inspired processes is further addressed in the next chapter where a hierarchical, developmental system is presented. This system takes the advantages of the formalisms discussed thus far as the basis for the design of a more flexible system, one that addresses the issues briefly touched on here.

8.5 The Developmental Algorithm

This section describes the algorithm used to simulate the rewriting and development of modules as time progresses. I will consider the context-free case, where the decision to rewrite a module will be based only on that module's age.

[61] For example, stance and swing phases as used in bipedal animation (Calvert 1991).

This section is primarily of practical interest, explaining how the formalisms presented in the first part of this chapter can be implemented as a computer program.

8.5.1 DOL-System Rewriting

For a discrete DOL-System, as defined in Section 5.3.2.1, rewriting is applied to a list of symbols, μ. Symbols are examined in parallel and matching productions from P are applied at each iteration to create the developmental sequence generated by the L-system G. The developmental sequence begins with the axiom, $\omega = \mu_0$, and the sequence proceeds $\mu_0 \Rightarrow \mu_1 \Rightarrow \ldots \Rightarrow \mu_n$ to a derivation of length n.

The rewriting process is illustrated graphically in Figure 8-23. Note the step-wise development of active symbols.

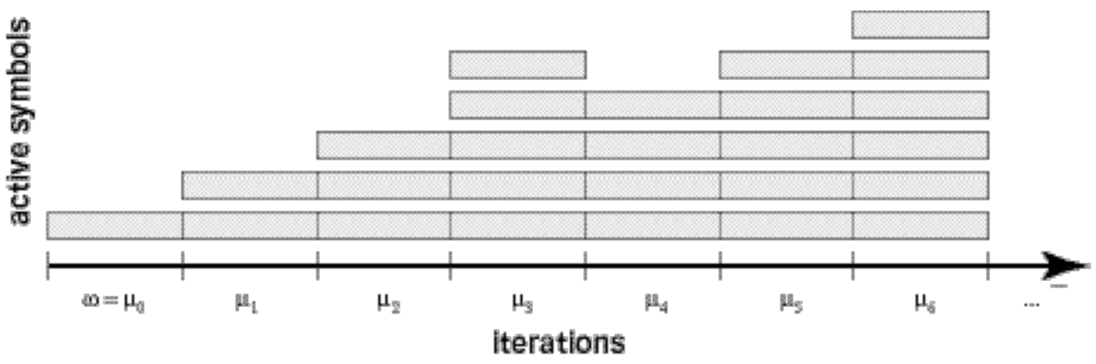


Figure 8-23: Iterative development of a DOL-system. Each grey box represents a symbol, so the current derivation word list is read vertically. At each iteration, the current word list is examined and the matched symbols are rewritten. Development proceeds from left to right, in discrete time steps.

The basic algorithm to implement this development is shown below in Figure 8-24.

```
/* Rewrite the supplied string - performed at each derivation step
*/
SymbolList RewriteSymbolList(SymbolList mu) {
   ModuleList muNew = empty SymbolList;
   for each symbol s in mu {
      /* check for a production for this symbol */
```

```
        if (RewritingProductionExists(s)) {
            /* if so, inset the new symbols into muNew */
            append (RewriteProduction(s)) into muNew;
        } else {
            /* otherwise, copy the symbol into muNew (identity
    production) */
            append (s) into muNew;
        }
    }
    return muNew;
}
```

Figure 8-24: Code fragment for string rewriting DOL-systems.

The algorithm shows the rewriting process for a single derivation step. A new list of symbols is generated by examining each symbol in turn and performing the rewriting if a production with predecessor `s` exists (handled by the `RewriteProduction` function). If no rewriting rule exists, the symbol `s` is copied into the new list (the identity production). The `RewriteSymbolList` function returns a new list, which is the derivation of the input list for a single iteration.

Normally this function is first called on the axiom and iteratively called for each derivation step as required. When the required derivation length is reached, the symbol list can be passed to the turtle interpreter to generate geometry from the current list of symbols.

8.5.2 Timed L-system Rewriting

In the case of timed L-systems described in Section 8.2, symbol rewriting is not as straightforward, because each symbol may be active for arbitrary amounts of time (Figure 8-25). Hence, symbol rewriting cannot proceed in the simple iterative mode of the previous algorithm. In this section, the algorithm used for context-free, stochastic, parametric, timed L-system derivation is described.

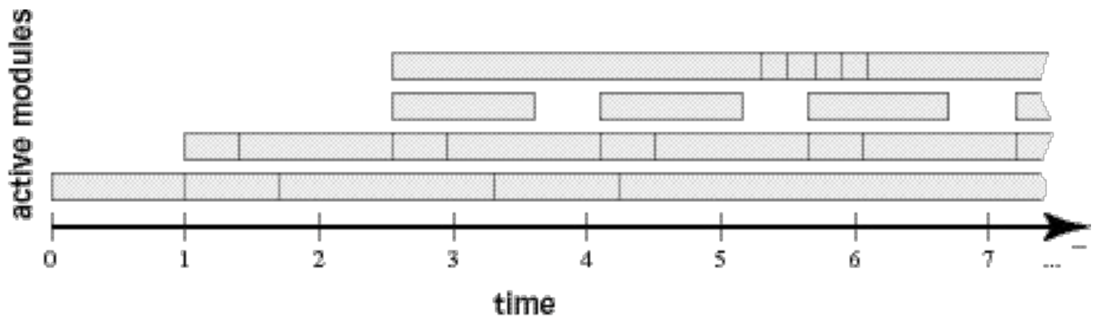


Figure 8-25: For timed L-system development, modules may be active for an arbitrary time period. The first rectangle at time 0 represents the axiom. Development proceeds as symbols age to their terminal age and are then rewritten. The discrete, step-wise development of DOL-systems no longer applies. (cf. Figure 8-23).

8.5.2.1 Overview of the Developmental Process

In describing this process, we assume a *global time* variable that is monotonically increasing. While development is theoretically a continuous process, in order to generate an animated sequence the system is sampled at discrete time intervals to create individual *frames*. Frames are played in sequence to give the illusion of animation. Frame sampling is independent of module development — the sample rate and time between samples is arbitrary and has no effect on module development. While the model described in Section 8.2 allows construction of a derivation string at any time, *t*, considerable practical savings in computation are possible if the derivation string is forward developed and information cached between samples, rather than computing the string from the axiom each time a new time sample is required. Once the system has developed to a particular global time using this caching scheme, it is not possible to sample the system at any previous time, without recomputing it from the axiom.

To sample the system, we require the state of the developing system at a specified future point in time $t_{i+1} = t_i + t_{i+}, t_{i+} > 0$, given that it has a current state (list of active modules) corresponding to the current global time, t_i. The *active module list* (developmental word) begins with instances of the modules that make up the axiom at their birth ages at a global time of 0.[62] Module development and the rewriting of modules proceeds as the current value of global time increases. This is a typical situation when creating an animated sequence.

[62] Birth ages may be expressions, and are evaluated for the axiom when global time is 0.

8.5.2.2 Mixing Timed and Non-Timed Modules

For practical purposes, not every symbol in the alphabet may need to undergo timed development. For example, the bracket symbols ("[" and "]") do not normally develop[63] in any sense and simply provide a mechanism for producing branching structures. For practical convenience, timed and non-timed symbols may be intermixed, in which case any rewriting proceeds at the normal integer derivation step of DOL-systems (the purpose of the `iLevel` component of `TimeStruct`, see below).

At any time, the active module list may contain modules of various ages. For context-free timed L-systems, a rewrite (application of a production) will occur when the current time equates to the terminal age of any symbol in the current list. If the state of the system is required at any time before the next rewrite event occurs, only the development functions need to be evaluated for the particular time the system is sampled. For non-timed modules, rewriting occurs if necessary each time the active module list is scanned.

8.5.2.3 Representing Time during Development

We assume the definition of a data structure, `TimeStruct`, which holds the necessary time information needed to keep track of development times as modules develop and productions are applied. This is shown as a C structure in Figure 8-26.

```
typedef struct {
   int ilevel;          /* current (integer) derivation length */
   S_age curTime;       /* current time (elapsed time) */
   S_age nextChange;    /* time until next production */
   S_age totalTime;     /* total time at this frame */
   S_age maxTime;       /* maximum time this system can run */
   int totaliLevel;     /* total integer level */
} TimeStruct;
```

Figure 8-26: the `TimeStruct` data structure.

[63] Meaning change their properties or parameter values over the lifetime of an individual module instance.

The S_age type represents the age of a symbol (i.e. the time it has existed — a floating-point number). The TimeStruct data structure stores relative times for a given L-system during its development. Its main purpose is to store the development time data of that particular system as productions are applied up to a required global time. The integer values are used to accommodate the non-timed components of a derivation string, showing the required (totaliLevel) and current derivation length (ilevel) for non-timed symbols. A sample scenario of the relationship between elements of TimeStruct and the developing L-system is shown in Figure 8-27.

8.5.2.4 Forward Prediction

The algorithm running the development of the timed modules needs to "forward predict" the time in the future when the next module rewrite will occur (which is the nextChange field in TimeStruct). Development between the time of the previous rewrite and the time of this next rewrite is continuous, in that (by definition) no rewriting of modules will occur, but the age of modules that are currently alive will advance (hence parameters affecting the module's turtle interpretation may vary). It is always possible that more than one module may be considered for rewriting at nextChange, which ensures the parallel rewriting nature of L-systems is maintained in this scheme.

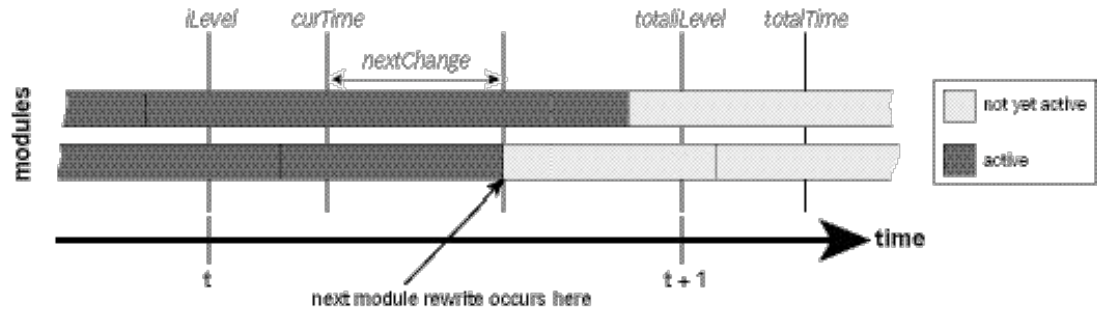


Figure 8-27: Sample instances of the TimeStruct members and their relation to the developing L-system modules. The dark grey modules are active at the current time, the variable nextChange determines the time remaining until the next rewrite occurs from the current time. The totalTime variable contains the time that development needs to proceed to, where upon active modules are supplied for turtle interpretation. The integer values iLevel and totaliLevel correspond to where un-timed symbol development proceeds from (they correspond to iterations in a DOL-system).

8.5.2.5 Developing Modules

The function DevelopModuleList takes a module list, a set of productions, an initialised TimeStruct and an evalTime and develops modules in the list from the interval over the current time (in the supplied TimeStruct) to evalTime. This function is shown below in Figure 8-28.

```
/* DevelopModuleList: */
ModuleList
DevelopModuleList(ModuleList mu, ProductionList p,
  TimeStruct *ts, S_age evalTime) {

   Boolean shouldGen = FALSE; /* used to flag generation */

   if (timeStruct->currentTime > evalTime) {
      print
    error "L-system is already developed past requested time"
      exit;
   }

   ts->totalTime = evalTime;
   ts->totaliLevel = floor(endTime);

   while ( (ts->nextChange < 0.0 && ts->ilevel < ts->totaliLevel) ||
      (ts->nextChange >= 0.0 && ts->nextChange + ts->curTime <
   evalTime)) {
      /* ApplyProductionSet controls development up to endTime
       * it also changes the values in ts (passed as a reference)
       * to reflect the state of development in mu when the function
       * returns.
       */
      mu = ApplyProductionSet(mu, p, ts, &shouldGen);
      if (shouldGen) {
      /* during the current application of productions we have
         found a need to interpret the current symbol list at
         this point. This is caused by the '%' symbol which
         stores the turtle reference frame in world coordinates
         at the time it is born.
         To achieve this we interpret the current symbol list
         without outputting any geometry. This is necessary to
         ensure the correct position and orientation of the
         turtle at the time and position the '%' symbol was
         encountered.
      */
```

```
            InterpretModuleList(TURTLE_ONLY, mu, ts->curTime);
        }
    }
        /* update the times to reflect the next symbol change
        and current development time
    */
    ts->nextChange -= (ts->totalTime - ts->curTime);
    ts->curTime = ts->totalTime;
    return mu;
}
```

Figure 8-28: Development of timed L-systems – the `DevelopModuleList` function.

The operation of `DevelopModuleList` is reasonably straightforward, the function returns a list of symbols developed up until the supplied end time, assuming that they are passed to the function in a state corresponding to some previous time (specified via the `TimeStruct` variable `ts`). In the case of creating an animation, `DevelopModuleList` is likely to be called every 25th or 30th of a second, depending on the frame rate.

The function `InterpretModuleList` directs the interpretation of the current list of modules by the turtle. It is included in the `DevelopModuleList` function to support the use of the "%" symbol (refer Section 8.3). No geometry is output or constructed within `DevelopModuleList`, only the turtle position, orientation and state is evaluated (ensured by the `TURTLE_ONLY` directive argument). The interpretation of the "%" module will include the saving of this state information.

Upon exit, `DevelopModuleList` has updated the `TimeStruct` variable `ts` to reflect the current state of the system's temporal development. The `nextChange` data member is updated by `ApplyProductionSet` and always reflects the closest future time that a module rewrite will occur. As productions are applied over the requested development period within `DevelopModuleList`'s while loop, the value of `nextChange` will update to indicate the time of the next necessary rewrite. This is possible because the terminal age of any module is set at its birth (i.e. its death is known in at the time of its instantiation).

8.5.2.6 Formal and Instance Modules

In this implementation, we distinguish between a *formal module* (Figure 8-1 on page 175), which contains the definition of a module (symbol name, formal parameters, terminal age) and its *instanced module* counterpart, which is the module generated as the system develops the active module list (derivation string). We will first look at the formal module.

```
/* Formal Module Definition */
typedef struct FormalModule {
   String moduleName;  /* text representation of module name */
   ParameterType parameters[];  /* list of formal parameters */
   Scalar defaults[];    /* default parameter values */
   S_var terminalAge;   /* terminal age for this module */
   Expression developmentFunc;    /* development function */
   Production productionList[];   /* this module's productions */
   int numProductions;   /* number of productions in list */
   struct FormalModule * equivs[];  /* list of equivelent modules */
   Boolean (*action)(...); /* action to perform when interpreting */
   .
   .
   .
} FormalModule;
```

Figure 8-29: Data structure showing key elements of a formal module.

As shown in Figure 8-30, the formal module includes the module's identifier (symbol name), a list of parameters and default values for each parameter. The number of defaults may be equal to or less than the number of formal parameters. Defaults are set for convenience, to allow unspecified parameter expressions in successor modules to assume defaults. In addition, for predefined symbols (e.g. "F"), it may be convenient to add additional parameters that are not interpreted by the turtle, but may play a role in the grammar. This makes productions such as:

$$\left(F\left(l,n\right),1\right): n>0 \rightarrow \left(F\left(l,0\right),1\right)\left(F\left(l/2,n-1\right),0\right)$$

possible, where both parameters of F are used in the production, but only the first parameter, l, is used in the turtle interpretation.

The terminal age of the module is of type `S_var`, which represents a scalar variable, capable of being used in expressions (it is commonly used in expressions related to the development function, described in Section 8.2.1). The formal module also includes data members to store this module's development function, list of productions, list of equivalent modules (Section 6.2), and the action to be performed by this module when interpreted by the turtle. Some additional parameters also form part of the formal module specification, these have been omitted in this discussion, as they are largely related to implementation issues and not important for the description here.

Instance modules are used in production successor lists. Each instance of a module stores a pointer to the corresponding formal definition of that module, a list of current parameter values, and associated timing information (shown in Figure 8-30 as a C struct).

```
/* Instance Module Definition */
typedef struct {
   FormalModule * formalModule;  /* pointer to the formal module
                                                    definition */
   Vector parameterValues;  /* current values of this module's
                                                    parameters */
   S_age birthday;   /* birth time for this element (real time) */
   S_age life; /* how long this instance of the module will live */
} Module;
```

Figure 8-30: Key elements of the data structure representing an instanced module.

The `birthday` field of the `Module` data structure is initialised with the current global time when this particular module was instantiated. The `life` field stores the relative lifetime of the module, so the terminal age of the module in terms of global time is `birthday + life`. The age of a particular module is calculated by the difference between `birthday` and the current global time.

8.5.2.7 Successor Modules and Productions

In order for the rewriting process to proceed, a particular production must be matched, and then the module being rewritten must be replaced with a series of instanced modules from the matched production. Figure 8-31 below shows the data structures used to represent productions and successor modules. The successor modules in a matched production are used to create the instanced modules in the derivation list.

```
/* Successor Module Definition */
typedef struct {
   FormalModule * formalModule;  /* pointer to the formal module */
   Expression expressionList[];  /* list of expressions to be
                                     evaluated  to create actual
                                     parameters for this successor
                                     module */
   Expression birthAge;       /* expression to evaluate birth age */
} SuccessorModule;

/* Production Definition */
typedef struct {
   Scalar probability; /* prob of application for this production */
   Expression predicate; /* condition to evaluate in order to apply
                               this production */
   SuccessorModule successors[]; /* list of successor modules */
} Production;
```

Figure 8-31: Data structures representing a successor module and production.

As the system permits stochastic matching in addition to logical predicate conditions, the `Production` data structure contains data members to allow matching. In the case of non-stochastic productions, the data member probability will be 1. For non-conditional productions, the predicate expression is set to `TRUE`. In the case where a production has a probability less than 1 *and* a predicate condition, the stochastic matching is tested first, if the rule is first matched stochastically, it is then tested for matching by evaluating the predicate (see Figure 8-32).

The `SuccessorModule` structure stores a pointer back to the formal module along with a list of expressions to be evaluated to calculate the actual parameters for the module. In addition, the expression `birthAge` is evaluated to calculate the module's birth age when instantiated into the derivation list (active module list). The birth age may of course be a constant, commonly 0. The members expressionList and birthAge correspond respectively to the expressions $e_1, e_2, \ldots, e_n$, and the α parameter in Figure 8-1 on page 175.

Formal modules, instanced modules and productions are used by the ApplyProductionSet function, detailed in the next section.

8.5.2.8 Development between Intervals

The real work in developing the timed L-system is done in the function ApplyProductionSet, which is detailed below in Figure 8-32. The utility function ModuleIsDead, which determines if a given Module has reached its terminal age, is also shown.

```
/* ModuleIsDead: returns TRUE if the supplied module is dead at
   the current time
*/
Boolean ModuleIsDead(Module * m, TimeStruct * ts) {
   return (m->life >= 0.0) &&
      ((ts->curTime — m->birthday) >= m->life);
}

/* AppleProductionSet: apply productions to a given time
 * This function performes timed rewriting and returns an updated
 * list of active modules.
 * It also updates the TimeStruct data structure to reflect the
 * advancement in development that has taken place within the
 * function. The Boolean variable should_gen is set to TRUE if
 * the ‘%’ symbol is encountered.
 */
ModuleList ApplyProductionSet(ModuleList mu,
                              ProductionList p,
                              TimeStruct * ts,
                              Boolean * should_gen) {

   ModuleList muNew; // new list of modules created by this function
   Scalar rnd, prTotal;
   S_age minChange = SCALAR_MAX, d;
   Boolean timed, nextTimed = FALSE, matched, shouldFree = FALSE;
   int i;

   (* should_gen) = FALSE;
   if ((timed = (ts->nextChange >= 0.0))) {
    /* move time forward to point of next change */
    ts->curTime += ts->nextChange;
}
repeat for each Module currentModule in mu {
    if (currentModule->formalModule->numProductions < 1) {
        if (!ModuleIsDead(currentModule, ts)) {
        /* there are no productions for this module and it is
            not dead, so the module is kept in the new list of
            current modules
        */
            insert currentModule into muNew;
        }
    } else { /* the current module has productions */
        if (!timed ||
            (timed && ts->curTime − currentModule->birthday >=
            (currentModule->life − MIN_CHANGE_EPS))) {
            rnd = UniformRandom(); /* generate a random number */
            matched = FALSE, prTotal = 0.0;
            repeat for all productions associated with this module {
                if (rnd − prTotal <= current production’s probability){
                /* for non-stochastic productions the probability will
                    always be 1, so the above condition will evaluate
                    to TRUE
                */
                    if (current production has a predicate) {
                        evaluate the current production’s predicate;
                        if (predicate evaluates to TRUE) {
                            matched = TRUE;
                            break;
                        }
                    } else {
                        matched = TRUE;
                        break;
                    }
                } else
                    prTotal += current production’s probability;
                }
                if (!matched) { /* no rules worked */
                    if (!timed) {
                        insert currentModule into muNew;

                    } /* timed */
            } else if (current production is not empty) {
                /* module has productions */
```

```
   repeat for all sucessor modules listed in the current production
{
      currentSuccessor = the current successor module;
         (* should_gen) =
            (*should_gen) || currentSuccessor == ‘%’;
      newModule = create a new instance module from
      currentSuccessor;
      initialise newModule;
      if (newModule has parameters) {
         evaluate newModule’s parameters from currentSuccessor’s
            parameter expressions;
      }
      if (currentSuccessor->birthAge) {
            /* if a birth age expression was supplied then
             * evaluate it, effectively setting a birth in the past
             */
         newModule->birthday = ts->curTime –
         evaluate(currentSuccessor->birthAge);
      } else {
         /* no birth age expression so it defaults to 0.0 */
         newModule->birthday = ts->currentTime;
      }
      if ((d = newModule->life) >= 0.0 &&
         (d +=  (newModule->birthday – ts->curTime)) < min_change) {
         if (d > MIN_CHANGE_EPSILON) {
            min_change = d, next_timed = TRUE;
         } else {
            newModule->birthday = ts->maxTime – newModule->life;
         }
      }
      should_free = TRUE;
   }
} else
   should_free = TRUE;
} else {
   if ((d = currentModule->life) >= 0.0 &&
      (d += (curModule->birthday – ts->curTime)) < minChange) {
      if (d > MIN_CHANGE_EPSILON)
      minChange = d, nextTimed = TRUE;
   else
      currentModule->birthday =
   ts->maxTime-currentModule->life;
}
if (newModule was created) {
   insert newModule into muNew;
```

```
    }
    if (should_free) {
        delete currentModule
    }

    if (next_timed) {
        /* update the TimeStruct's nextChange member to reflect the
         * time of the next timed module rewrite and check for
         * consistency (change happens in the future).
         */
        if ((ts->nextChange = minChange) <= 0.0) {
            print error "Next change has already happened" and exit;
        }
    } else
    ts->nextChange = -1.0;

    ts->ilevel++; /* increment integer derivation length */
    return muNew;
}
```

Figure 8-32: The `ApplyProductionSet` function, which controls the development of timed L-systems.

The constant `MIN_CHANGE_EPSILON` is used for both efficiency and accuracy. In a complex development sequence, many modules may have times for rewrites that are very close to each other but not at *exactly* the same time. Moreover, the limits of accuracy of machine representation of real numbers and rounding errors may cause small inaccuracies in the calculation of module terminal ages. Any modules that the system predicts are due for rewriting within this small tolerance (`MIN_CHANGE_EPSILON`) are dealt with as a parallel production (i.e. all rewritten at the same time). This situation is illustrated below in Figure 8-33. Typically, `MIN_CHANGE_EPSILON` is kept very small — within 1–2 orders of machine precision (around 10^{-7} for 32 bit floating point representation).

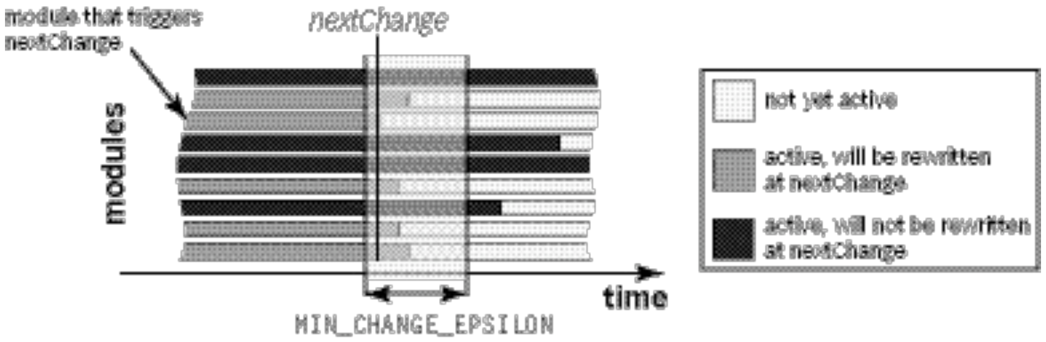

module that triggers nextChange
nextChange
modules
time
MIN_CHANGE_EPSILON
not yet active
active, will be rewritten at nextChange
active, will not be rewritten at nextChange

Figure 8-33: The `MIN_CHANGE_EPSILON` tolerance constant treats changes within a small range as being applied in parallel.

8.6 Discussion

The key advantage of the algorithm for animation purposes is the ability to keep the sample rate and module development separate. This allows the same L-system to be sampled at any point in time, with the guarantee that the state of the system will be accurate and consistent, independent of the number of times, or the intervals between which the system has been sampled previously. For example, it is often necessary to sample at different frame rates when creating animation for film (24 fps[64]) or video (25 fps or 30 fps). This method permits sampling at any interval greater than the `MIN_CHANGE_EPSILON` constant. This is possible due to the constraint that the only way a module can be rewritten is when it reaches its terminal age — which is known at the time of the module's birth. The use of an incremental development algorithm does not require the recalculation of developmental words from the axiom at each positive incremental time sample, resulting in increased efficiency.

However, these advantages can also be limitations, particularly when we are trying to model other classes of developmental systems, such as reaction-diffusion systems, which require associated continuous differential development, where module terminal ages cannot be fixed at birth. In addition, systems are required where real-time, environmental, or context information influences development. In this situation the prediction ability of the algorithm (described in Section 8.5.2.4), breaks down due to the non-determinacy that these additional components add.

[64] Frames per second — the frame rate.

One additional problem comes in designing a system that incorporates hierarchical organization, complex structure and development. The grammar-based nature of L-systems makes them suited to capturing hierarchical structure but by incorporating the timed developmental formalisms described in this chapter, the distinction between structure and development is blurred, hence designing very complex models becomes difficult.

The work described in the next section, addresses these limitations by adding the ability to include environmental and context information. Importantly, the system also includes the practical ability to run and respond to environmental input in real-time, making it suitable for applications such as music and interactive animation generation. Modules no longer contain a fixed terminal age, the decision for a module to terminate or be rewritten being subject to a wider range of conditions.

9 A Developmental System for Generative Media

Growth of one part must affect growth of all parts. The rules are rigorous, but within those rules variety abounds, and the rules show through the variations to portray a relatedness of parts that is aesthetically pleasing and a consistency of purpose that provides an eternal model for all of man's creations.

—Peter S. Stevens (Stevens 1974, page 222)

Taking the formalisms described in the previous chapters, I now proceed to extend the model to include additional developmental components and a full hierarchical description capability. Here I depart from the traditional L-systems description, as it is easier to consider the developmental and implementation details using a slightly different terminology.

This chapter describes a practical system that builds on the theoretical and practical models described in previous chapters. Fundamentally, this model incorporates elements from hierarchical, timed, context sensitive, deterministic and differential L-systems.

The emphasis of this chapter is on the design and construction of a practical programming system for modelling organic forms and processes. The goal of the system described is that of a flexible generative system, capable of generating dynamic models in a variety of different domains (e.g. graphics and animation, biological and botanical modelling, music composition, and real-time, interactive animation).

9.1 Related Work

The extensions described in this chapter bring together a number of related developments in L-systems, morphogenesis modelling, object-oriented programming and dynamic media. These include L-systems that incorporate development via differential equations (Prusinkiewicz, Hammel & Mjolsness 1993), the sub-L-systems of Hanan (Hanan 1992), the cellular developmental system of Fleischer et. al. (Fleischer & Barr 1994; Fleischer et al. 1995) and the MET++ application framework of Ackermann (Ackermann 1996). Relationships between previous work and the system described here will be expanded in the sections related to those specific areas that draw upon previous work.

9.2 The Developmental Model

9.2.1 Cell Definitions

In developing this system, we will consider a basic automaton, which is referred to as a *cell*. The name is used as a metaphorical interpretation of cells as found in biological life. The model is "biologically inspired", but is not designed to reflect a literal interpretation of cellular development. This cellular abstraction is capable (as a simulation) of functions a biological cell does not have, and reciprocally, the biological cell is capable of many functions not possible with the model described here. Cells exist in an abstract entity called the *world*. The world is responsible for the *creation, removal,* and *interpretation* of cells that exist within it. Details of these terms and the world itself will be discussed shortly.

A cell is composed of four principle components (refer Figure 9-1):

- A *label,* $s \in V_T$, where V_T is an alphabet that is specific to the cell *type* (corresponding to the alphabet of an L-system). The type of a cell is distinguished by its ability to develop or be interpreted (Section 9.2.4).
- A *state,* $\Sigma_T \in \left(\Re^* \times \Im^*\right)$ — a set of variables that reflect measurable properties (both internal and external) that the cell possesses. The state will change dynamically subject to the mechanisms of the cell (detailed in Section 9.2.2).

- A set of predicate *rules* or *productions,* $P_T \subset \left(V \times \Sigma^*\right) \times C(\Sigma) \times \left(V \times E(\Sigma)^*\right)^*$ that specify developmental changes to the cell. These productions may consider the cell state as well as the state of *neighbouring* cells.
- An *interpretation,* $I \subset \left(V_I \times E(\Sigma_T)\right)^*$, which is a set of instructions as to how the cell is to be realised in the world (V_I is the alphabet of a particular set of interpretative symbols, see Section 9.2.4). The interpretation can make use of the cell's state.

Cellular instantiation follows the class/object model used in object-oriented programming (Wirfs-Brock, Wilkerson & Wiener 1990).[65] Cells are instantiated into pools (explained below), and there may be many *instances* cells with the same label, but each cell carries its own state, which develops independently. Conceptually, each cell also carries its own copy of the rules and interpretation defined for a cell of that label, although in most situations these are references to the rules and interpretation contained in the *master cell*. Thus, normally no distinction needs to be made between master cells and instance cells.

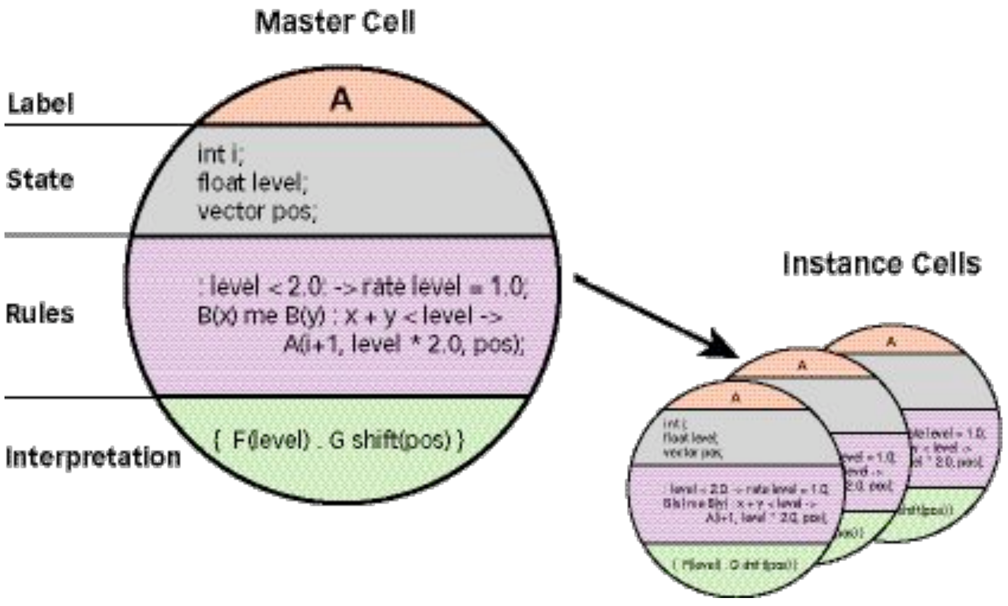

Master Cell
Label
State
Rules
Interpretation
A
int i;
float level;
vector pos;
: level < 2.0: -> rate level = 1.0;
B(x) me B(y) : x + y < level ->
A(i+1, level * 2.0, pos);
{ F(level) . G shift(pos) }
Instance Cells

Figure 9-1: Master and Instance cells, and their principle components.

A special type of cell is called a *system*. A system has a label and contains state information, but does not have any rules or interpretation. Unlike a

[65] Alan Kay, inventor of the *Smalltalk* programming language, used biological metaphors in its design, likening the concept of objects to "cells" with walls, with the class providing a well-defined boundary between co-operating units (Kay 1993).

normal cell, a system cell may contain other cells, including other system cells. Thus, the system cell is capable of forming a *hierarchical structure*. Systems contain an *initial state* (or *axiom*) that consists of a sequence of instance cells with particular state initialisation information. They also maintain a *pool* (spatial data structure) wherein cells may be created, replaced, and deleted. A *root system* contains all other cells and systems and is created automatically by the world upon initialisation. The root system's age will automatically reflect the developmental time of the entire system.

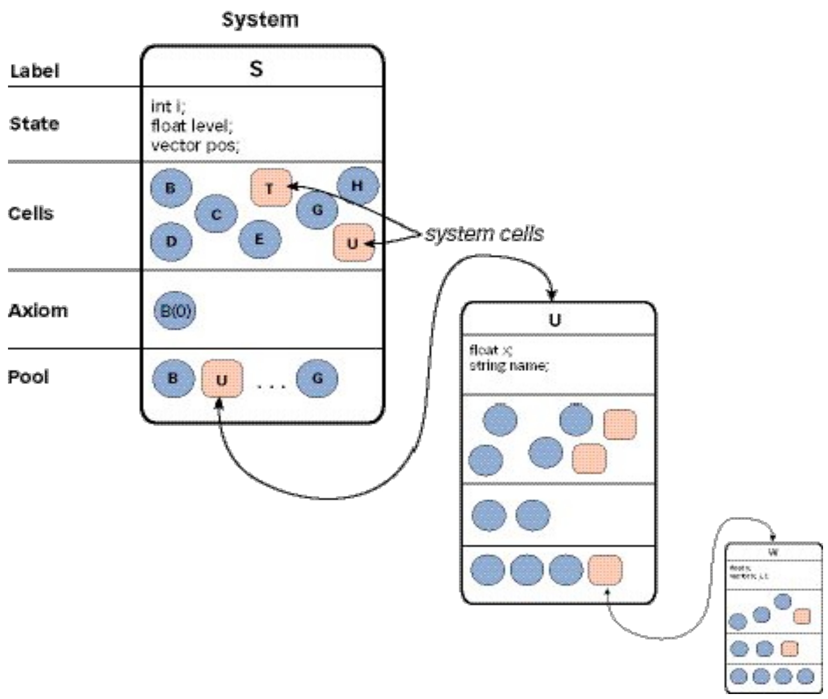


Figure 9-2: A system cell may contain other cells which in turn may be system cells. Thus the cellular hierarchy is formed.

System cells may contain other cells (which may be systems too), but they cannot contain instances of themselves, nor can sub-systems contain instances of parent cells. This ensures the cellular hierarchy maintains a tree structure (rather than a cyclic graph). A hierarchical structure is a good way of describing many natural patterns and forms (Simon 1996).

Cells within a system develop asynchronously, however cells *may* synchronise development based on examination of each other's state. In addition, cells have access to the state of any parent cells, including the state of the root system.

Specific components of cells will now be described in more detail.

9.2.2 Cell State

Cell state captures the measurable components of a cell. The state is a vector composed of both user- and system-defined variables. These variables may be of type *integer, real, vector* (a scalar triplet) and *string*. The user may define the cell state as required by the cell's particular type. In addition, all cells maintain a number of internal states that are defined and managed by the cell itself. Internal states are "read-only" — available as symbols for use in productions, but they cannot be modified, hence they provide an introspection of various fixed components of the cell. The internal state includes the cell *age,* which is automatically updated to reflect the age of the cell during its lifetime. An internal *status* is also maintained, which has four potential values affecting overall cell behaviour:

1. ***Dormant***: the cell is effectively inactive. There is no dynamic state management, the cell does not age and no rules are applied. Master cells are normally kept in the dormant state.
2. ***Birth***: an initialisation status that allows the user defined section of the state vector to be initialised. Here the age of the cell is also set to its initial value (usually 0). Cells with this status will automatically be added to the system cell's pool.
3. ***Alive***: the normal status for an active cell. The cell dynamically manages its state via the application of rules.
4. ***Dead***: the status is set to this value when certain conditions are met, such as a *replacement* or *delete* action (detailed in Section 9.2.3.4). Dead cells in a system cell's pool are automatically removed from the pool.

The cell status can only change in positive sequence, i.e. $1(Dormant) \rightarrow 2(Birth) \rightarrow 3(Alive) \rightarrow 4(Dead)$, cells always begin with status 1 (dormant) and end with status 4 (dead). Dead cells are removed automatically (corresponding to a form of automatic garbage collection used in programming languages). The *age* state variable is reset at birth and continues to increment as long as the cell is in state 3.

9.2.2.1 *Scope and Access*

The individual elements of the state vector may belong to one of two possible classes: *internal* and *public*. Internal class elements are accessible within the cell only (they are local to the cell). Public elements may be accessed by other cells within the same parent system and the parent system itself. In addition, public elements of the parent system's state can be accessed by child cells (including system cells and their child cells). These scope rules are similar to those for procedural programming languages, such as Pascal, that permit nested procedure descriptions.

Systems and modules all have a unique identifier depending on their position in the hierarchy. Modules from different systems may appear to have the same name due to scope rules, which allow for modules in different systems to appear to have the same name. For example:

```
system s1 {
  module A() {
   rules:
     : age > timestep : -> A() s2;
  }
} A();
system s2 {
  module A() {
   rules:
     : age > timestep : -> s1;
  }
} A();
```

The fully qualified name of module `A` in `s1` is `root::s1::A`, where as the module `A` in `s2` is `root::s2::A` (recall that the root system is the parent of all systems). Full qualification of scope is usually unnecessary, as the internal symbol table searches for matching identifier from the current scope outwards. Hence for the rule for `s1::A`, the `A()` in the cell action matches the current module scope where as `s2` matches the next level (system) scope.

9.2.2.2 *State Initialisation*

When a cells status changes from status 1 (Dormant) to 2 (Birth) its state is initialised and the cell begins to age incrementally. If the cell is a system cell, state initialisation includes the instantiation of cells in the systems axiom. This may propagate initialisation changes further down the hierarchy, since axiom cells of one system may also be system cells with their own axioms. As the hierarchical structure is acyclic, the initialisation process is guaranteed to terminate.

9.2.3 Cell Rules

Cellular rules, denoted r_i are ordered sets of *predicate-action* sequences of the form:

$$r_i : \underbrace{\{context\} : predicate}_{\text{predicate component}} : \underbrace{(state\ calculations \mid cell\ actions)}_{\text{action component}} \tag{9.1}$$

Rules are numbered implicitly in the order of their declaration. While a cell is in the *alive* state, its set of rules is evaluated in ascending order. If the predicate component evaluates to `TRUE` (non-zero), the action component is executed. The action component may consist of calculations that change the cell state, or actions that the cell should perform. The following sections describe each component in more detail.

9.2.3.1 *Context*

A context statement involves the position in the pool of the current cell in relation to other cells. A special reserved word, `me`, is used to represent the current cell. This allows cells with the same label to be used in context specifications. Context statements also specify the public state variables of cells involved in the context specification, and these identifiers may then be used in state calculations involving the current cell, i.e. a cell may update its state based on the state of its neighbours. Access to the state of neighbouring cells is read-only. Changing another cell's state directly is not permitted.

Here is an example context sensitive rule, where a cell maintains a component of its state to be the average of its neighbours[66], provided it exceeds some minimum threshold:

$$A(y)\ me\ B(z) : y > k_{\min}\ \&\&\ z > k_{\min} : x = \frac{y+z}{2}$$

which assumes the cell that owns this rule has a state variable, x. The rule first checks if the context is satisfied — that cells with labels "A" and "B" are at the "left" and "right" of the current cell. If that relationship is TRUE, the state parameters are then checked to see if they exceed some minimum constant value ($k_{\min}$), if so the current cell's state variable x is updated to be the average of the values of it's neighbours.

9.2.3.1.1 The Meaning of Context Relations

Context relations have a more flexible meaning than with the context sensitive L-systems discussed in Section 5.3.3. In the case of L-systems, the derivation string is a one-dimensional array, and so context matches are decided on by matching symbols to the left and right of the current symbol in the derivation string (a one-dimensional context). The pool in which the cells exist is designed in an abstract way, where the interpretation of neighbour relationships is flexible. This is achieved using polymorphic functions to match context based on pool type. It is important to match context dimension to pool dimension (e.g., a two-dimensional context relation makes no sense to a one-dimensional array).

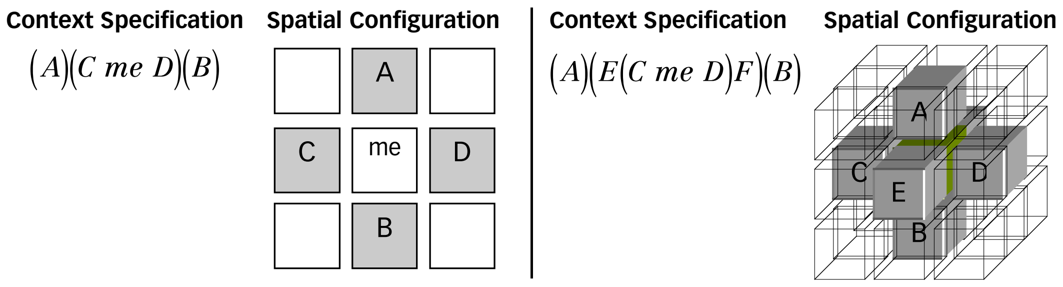


Figure 9-3: Higher dimensional context relations and their specification.

In the cellular developmental system, context may include other *spatial* relationships where context relations are satisfied when the spatial position of the

[66] This example uses a one-dimensional context relation; higher dimensional relations are defined in the next section.

cell is less than some Euclidian distance, or topological relationships such as the Von Neumann neighbourhood (Figure 9-3) used in cellular automata simulations (Berlekamp, Conway & Guy 1982). The use of parenthesis demarks dimensions when specifying context.

As the system described here uses polymorphic objects to represent the pool (e.g. linear set, multi-dimensional array, spatial structure), the interpretation of context is determined by the way the specific pool interprets the context statements (refer Section 9.6.1.4). This permits flexibility in the types of simulations the system can perform. For example, such context relationships can be used in music (see Chapter 11), where context relations work in two dimensions: pitch and time (hence context matching can be with chords, rather than notes, see Section 11.6.3).

9.2.3.1.2 Context Example: Reaction-Diffusion Systems

As an example of the use of context sensitivity in higher dimensions, consider the process of ***reaction-diffusion,*** originally proposed by Alan Turing as a chemical basis for morphogenesis (Turing 1952). Reaction-diffusion systems have found application in computer graphics research in the simulation of natural surface textures such as animal spots and stripes (Turk 1991; Witkin & Kass 1991). A simple, two-chemical system can be modelled with the following equations:

$$\begin{aligned}\frac{\partial a}{\partial t} &= F(a,b) + D_a \nabla^2 a \\ \frac{\partial b}{\partial t} &= G(a,b) + D_b \nabla^2 b\end{aligned} \tag{9.2}$$

Where a and b are the concentrations of two chemicals, known as the *inhibitor* and *activator*, t is time, D_a and D_b the diffusion rates of a and b respectively. The Laplacian ∇^2 represents the local concentration gradient, and the functions F and G are the reaction functions for the two chemicals, determining the rate of chemical production. Turk (Turk 1991) shows how Turing's reaction-diffusion model can be simulated in discrete form on a two-dimensional cellular grid, using the Von Neumann neighbourhood:

$$\begin{aligned}\Delta a_{i,j} &= s\left(16 - a_{i,j}b_{i,j}\right) + D_a\left(a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1} - 4a_{i,j}\right) \\ \Delta b_{i,j} &= s\left(a_{i,j}b_{i,j} - b_{i,j} - \beta_{i,j}\right) + D_b\left(b_{i+1,j} + b_{i-1,j} + b_{i,j+1} + b_{i,j-1} - 4b_{i,j}\right)\end{aligned} \tag{9.3}$$

Where s is a scaling coefficient, and β a random variate used to account for slight irregularities in cellular concentration (giving the texture a more "natural" feel). Running this model over a two-dimensional cell grid produces characteristic reaction-diffusion spot patterns. We can simulate the reaction-diffusion using the cellular developmental model described in this chapter using context sensitivity.

```
// simple 2D reaction diffusion cellular system
system ReactionDiffusion(integer x, integer y) {
   real DA = 0.125;
   real Db = 0.0325;
   real s = 0.02;
   module R(real a, real b) { // a single r-d cell
      real beta = 12 + random(-0.2, 0.2);
      a = b = 4.0;
    rules:
            (R(a_ij1,b_ij1))
   (R(a_i_1j,b_i_1j) me R(a_i1j,b_i1j))
            (R(a_ij_1,b_ij_1))
   : TRUE : rate a = s*(16-ab)+Da*(a_i1j+a+i_1j+a_ij1+a+ij_1-4*a),
         rate b = s*(a*b-b-beta)+Db*(b_i1j+b_i_1j+b_ij1+b_ij_1-4*b);
   geom:
   // convert activator and inhibitor concentrations to hsv colour
      hsv(180*(linearstep(0,4,a) - linearstep(0,4,b)),0.5,
         linearstep(0,8,a+b));
   } // module R
   .
   .
   .
```

Figure 9-4: A 2D reaction diffusion cell system.

The cellular specification shown above needs only a single cell specification (R) to carry out the reaction-diffusion process (as it operates in the same way for each cell). For clarity, the modules required to construct the cell grid have been omitted, the modules construct the cells when the ReactionDiffusion system's axiom is run, but then play no further part in the simulation.

Module `R` has two public state variables, `a` and `b`, representing the concentration of inhibitor and activator morphogens in the cell. The differential `rate` operator updates the concentrations according to the Turing model, using the concentrations of neighbouring cells based on the context relation specified in the cell rule (the implementation details of context matching are described in Section 9.6.1.4). Finally, the concentrations are converted to colour information, modulating hue and brightness of colour based on the amounts of activator and inhibitor chemicals. The flexibility of the geom specification means that the chemical concentrations could be used to modulate other properties, either geometric (e.g. Figure 7-3) or even musical. The textural output of the reaction-diffusion simulation is shown below in Figure 9-5.

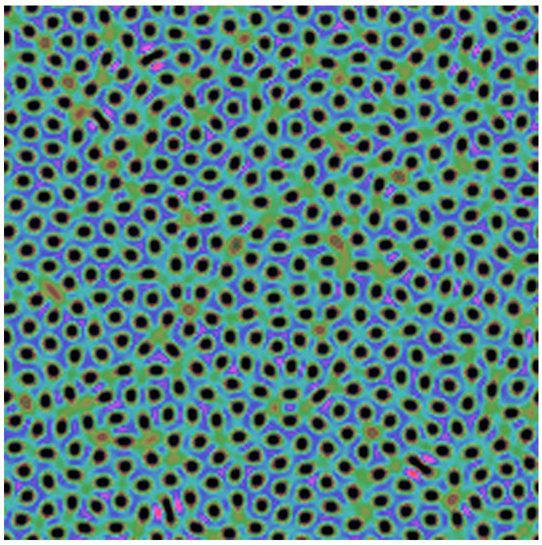


Figure 9-5: Reaction-diffusion simulation output.

9.2.3.2 Predicates

Predicates are mathematical expressions that evaluate to Boolean outcomes: TRUE or FALSE. Predicates may be composed of constants, global variables (such as the current time or frame), state variables of the current cell or publicly accessible state variables of neighbouring cells, in addition to standard logical and mathematical operators, plus the functions detailed in Section 6.8. Any expression that evaluates to 0 is assumed FALSE, any other value is TRUE. Null or empty strings are automatically cast to the integer value of 0.

9.2.3.3 State Calculations and the differential operator

State calculations are mathematical expressions that affect a cell's state. They are expressed in a similar manner to expressions in the C programming language (Kernighan & Ritchie 1988). Functions are the same as those detailed for parametric L-systems described in Section 6.8. There is one important addition, the unary differential operator, `rate`, which performs a specific form of ordinary differential equation solving with initial values. The operand for rate is a local cell state variable. Typically, the initial value problem is expressed,

$$x' = \frac{dx}{dt} = f(t, x), \quad 0 \le t \le \tau, x(0) = x_0 \tag{9.4}$$

where t represents the relative age of the cell and x_0 the initial value of the state variable x.

For example, consider a cell with state variable x declared,

$$real \quad x = 2.0$$

specifies the value on birth for the cell (the initial value or boundary condition). The rule

$$: x > X_{\min} : rate\ x = k * x, \tag{9.5}$$

is equivalent to the differential equation

$$\frac{dx}{dt} = kx, \quad x(0) = 2, x(t) > X_{\min} \tag{9.6}$$

where k and $X_{\min}$ are constants and t represents the cell's age relative to its birth. Such a function could represent the decay of a chemical concentration within the cell.

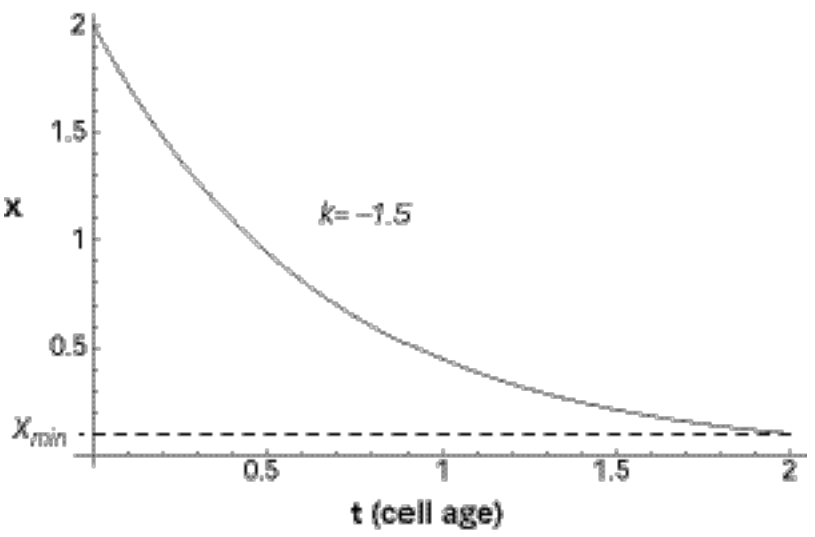


Figure 9-6: Graph of the state variable x over the first 2 time units of the life of the cell.

At the point where $x = X_{\min}$ an *event* occurs where the predicate changes state.

Higher order differential equations can be specified by multiple uses of the rate operator, e.g.

$$rate\ rate\ x = rate\ y - k$$

is equivalent to the mathematical expression

$$\frac{d^2x}{dt^2} = \frac{dy}{dt} - k$$

All state variables are assumed independent. Differential equations can be used to simulate simple processes both within and external to a cell (they can be based on public states of neighbouring cells for example). The state vector is updated using numerical integration methods at each time step.

9.2.3.4 Actions

Actions correspond to the re-writing process in L-systems, being similar to the successor word of L-systems. However, the concept of rewriting can be misleading, due to the way cell actions are handled and cells are placed in the pool. In the case of L-systems, symbols are always *replaced* by rewriting productions. Whereas, for the cellular developmental model, in addition to replacement cells may continue to exist in the pool while their actions cause new cells to be added (i.e. the cell action does not necessarily replace (rewrite) the cell instigating that action). Possible actions are outlined in Table 9–1.

The rules for creation and replacement (2 & 3) permit a context specification in the action, the syntax being the same as for context sensitive predicates (Section 9.2.3.1).

Type	Description	Example Syntax of the Action
1.	No action (the current cell remains)	$\rightarrow me$
2.	The current cell is replaced	$\rightarrow A(x,y)$
3.	New cells are created and the current cell remains in the pool.	$\rightarrow A(x,y)\ me\ B(x+y)$
4.	The current cell is deleted	$\rightarrow \varnothing$ (empty string)

Table 9–1: Rule actions.

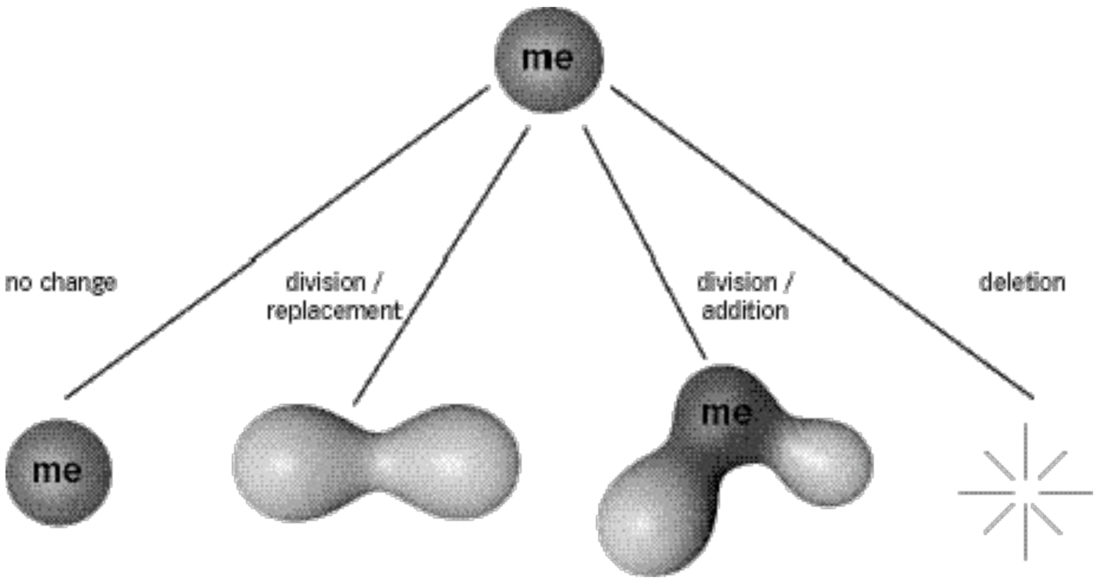


Figure 9-7: Graphical representation of cell changes brought about by rule actions.

9.2.4 Cell Interpretation

Thus far, cells have been considered in abstraction, without any method of realising them in the world. The interpretation component specifies how the cell will be interpreted as the system develops. In the case of L-systems, developmental words are interpreted by a turtle, which creates geometry as it interprets the list of symbols. The case here is similar, with the exception that each cell may contain multiple instructions to the turtle, permitting individual cells to create more complex geometry without polluting the cellular developmental model with cells used only as part of some complex geometry building sequence.

The interpretation system is extremely flexible, in the sense that interpretation contains a list of special cells representing instructions. These cells are of a different type than normal developing cells (they do not develop), but may have associated parameters. These parameters may be set with expressions involving the cell's state. If a cell's state is changing over the lifetime of that cell (the *age* component for example), then the interpretation permits the ability to animate the parameters of interpretive instructions as the cell state changes. This is a more flexible and general form of the development functions associated with timed L-systems and described in Section 8.2.1. There is no direct dependence between cell development and interpretation, so a number of different interpretative sets can be used on the same developmental system. For example, a musical interpretative set issues musical instructions rather than geometric building ones.

This flexibility allows users of the system to realise their developmental system in a variety of ways, without the need to completely respecify the grammar. The use of multiple instruction sequences in a single cell is a different solution to a similar problem encountered by Prusinkiewicz and colleagues in the development of their interactive system to model plants (Prusinkiewicz et al. 2001). Here they combined a C-like programming language and Chomsky grammars to enable sequential rewriting of strings, rather than the parallel development specified by L-systems.

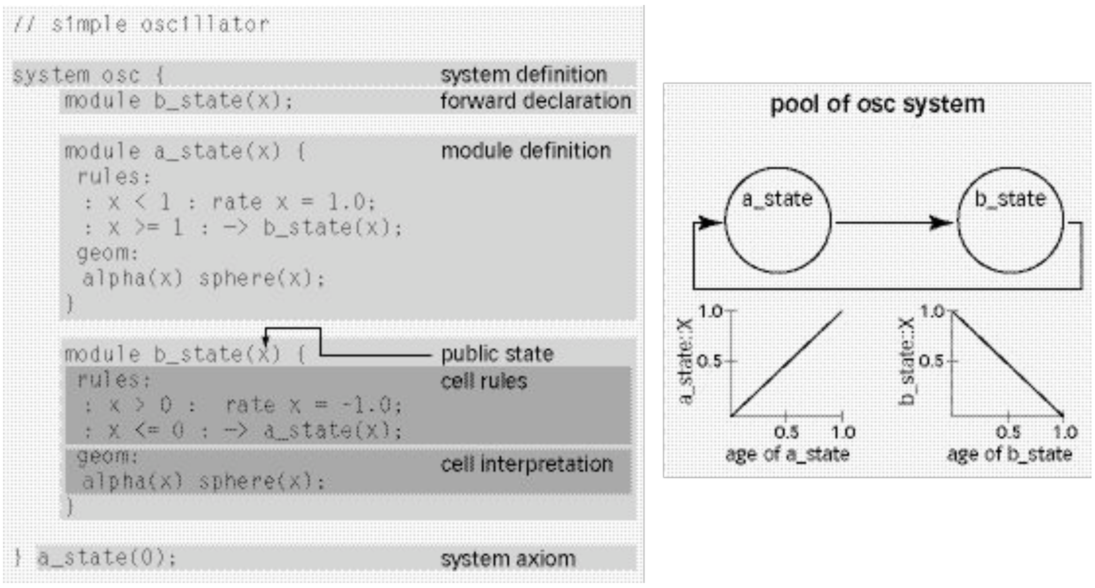
9.3 Cell Programming

The configuration of the system can be specified via a human-readable specification. The basic syntax of this specification is discussed here and is used in the description of examples in the next section.

Cells are specified using the keywords `module` and `system`, representing normal and system cells respectively. Modules of a particular system may be forward declared if required.

9.3.1 A 2-State Oscillator Example

Consider the example of a system that oscillates continuously between two states. Such a system could be used, for example, as part of a generative audio system, however for illustrative purposes a graphical interpretation will be used.



```
// simple oscillator

system osc {
    module b_state(x);

    module a_state(x) {
     rules:
      : x < 1 : rate x = 1.0;
      : x >= 1 : -> b_state(x);
     geom:
      alpha(x) sphere(x);
    }

    module b_state(x) {
     rules:
      : x > 0 :  rate x = -1.0;
      : x <= 0 : -> a_state(x);
     geom:
      alpha(x) sphere(x);
    }

} a_state(0);
```

Figure 9-8: Simple program for an oscillator. The pool of the system defined is shown on the right. It oscillates between two cells representing different stages of the oscillation. The state variable of each increases or decreases at a constant rate, forming a continuous oscillator.

The example shown above illustrates the key concepts of a specification. A single system, `osc` contains two modules `a_state` and `b_state`. The publicly accessible state variables of each module are specified in parenthesis following the declaration of the module's name. Any further information inside the opening "{" character is considered private state declaration and state initialisation expressions. Statements are terminated with a semicolon (";"). The keyword `rules:` prefixes the definition of rules for the module. Each rule is likewise terminated with a semicolon and may span multiple lines if necessary. The module's interpretation commands are prefixed with the `geom:` keyword, although there is no reason the interpretation has to be geometric. The commands `alpha` and `sphere` set the drawing transparency and create a sphere respectively. The supplied parameter controls the amount of transparency (in the case of `alpha`) and the radius of the sphere. In this case, the parameter provides a visual indication of the state of the module.

When the system `osc` is instantiated, it loads the pool with the axiom (`a_state`), initialising the state to the value specified. The system then begins development for as long as necessary. Figure 9-9 shows a sequence of frames outlining the development of the oscillator system.

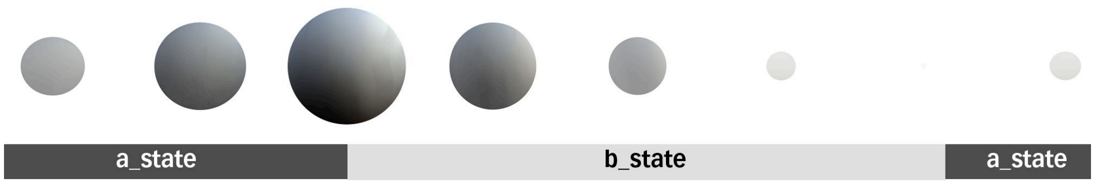


Figure 9-9: Sequence of frames from the rendered output of the oscillator system.

9.3.2 Hierarchy of Legged Figures

The animation of legged figures was presented in Section 8.4. I will now revisit this definition and show how the same system can be defined using the cell developmental model introduced in this chapter.

Complex structures are often modelled by decomposing them into a hierarchy. The running animal model presented in the previous chapter used a single L-system to model the structure and behaviour. By using a hierarchical description, it is possible to specify structure in a more intuitive way.

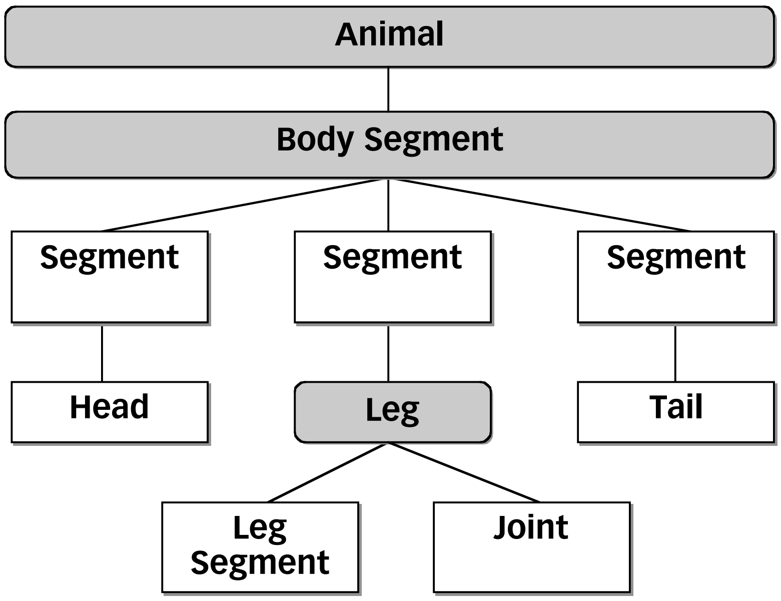


Figure 9-10: Hierarchical structure of the legged creature. Systems are shaded in grey.

A skeletal description of this structure using the cellular programming language is shown below. Elements that construct the geometry or control the legged gaits themselves have been omitted for readability (the legged gait animation system was described in Section 8.4).

```
system animal(vector position) {
   int nSegs = 4;
   system bodySegment(int n) {
      module segment(int n) {
         real kphase = 2 * PI/nSegs;
       rules:
         : n <= 0 : -> head segment(n+1);
         : n > 0 && n < nSegs : -> leg(n * kphase) ( segment(n+1) )
                                 leg((n+0.5) * kphase);
         : n >= nSegs : -> tail;
       geom:
         // body segment geometry defined here

      } // segment

      module head() {
         ... // constructs the head
      }

      module tail() {
         ... // constructs the tail
      }

      system leg(real phase, real ctime) {
         module joint(real phase, real ctime, int dof) {
            ... // joint articulation
         }

         module legSegment(real scale) {
            ... // constructs the leg segment (limb)
         }
      } joint(phase,ctime,3) legSegment(1) joint(phase,ctime,1)
        legSegment(1.5); // axiom for leg system
   } segment(n); // axiom for system bodySegment
 geom:
   TP(position); // set absolute turtle position
} bodySegment(0); // axiom for animal system
```

Figure 9-11: Specification of systems and modules corresponding to the hierarchy illustrated in Figure 9-10.

The rules for the module `segment` illustrate how specific context relations can be used in productions. Specifically, the rule

```
        : n > 0 && n < nSegs : -> leg(n * kphase) ( segment(n+1) )
                                   leg((n+0.5) * kphase);
```

places the animated legs at either side of the body segment, giving a bilateral symmetry, the phase of the gait shifted between legs.

9.3.3 Relation to Sub-L-systems

Sub-L-systems, first proposed by Hanan (Hanan 1992) allow different sets of rules to be applied to portions of the developmental string. Hanan also incorporated the ability to run sub-L-systems at different scales and time steps. This mechanism was introduced to enable the integration of previously defined components, yielding a hierarchical model with nested sub-L-systems.

This section briefly shows how sub-L-systems can be simulated into the cellular developmental model introduced in this chapter. Consider the following sub-L-system, taken from (Hanan 1992, p. 107). The sub-L-system simulates a simple branching structure.

$$\begin{array}{rl} \text{Lsystem}: & 1\ \ /*\ \text{Main L-system}\ */ \\ \omega: & A \\ p_{11}: & A \rightarrow I\left[?(2)A\$\right]A \\ & \text{endLsystem} \\ \\ \text{Lsystem}: & 2\ /*\ \text{sub-L-system for the branch}\ */ \\ \omega: & A \\ p_{21}: & A \rightarrow IA \\ & \text{endLsystem} \end{array} \tag{9.7}$$

The equivalent cellular model is shown in Figure 9-12.

```
module I() { }
system s1() {
   module A() {
    rules:
      :age % timestep == 0: -> I [ s2 ] me;
   }
} A();

system s2() {
   module A() {
    rules:
      :age % timestep == 0: -> I me;
   }
} A();
```

Figure 9-12: Cellular definition of the sub-L-system.

There are some differences between the two models. Firstly, as the cellular system is inherently time-based discrete symbol rewrites must be simulated using module age and the build-in global variable `timestep`. By varying time steps (via a parameter for example) other systems can develop at different rates. Secondly, sub-L-systems operate different replacement rules on the same symbols, however in this example there are two `A` modules. Their `geom` section could be made the same, giving them the same interpretation however. The sequence of development over the first five time steps is:

s1::A

I [s2::A] s1::A

I [I s2::A] I [s2::A] s1::A

I [I I s2::A] I [I s2::A] I [s2::A] s1::A

I [I I I s2::A] I [I I s2::A] I [I s2::A] I [s2::A] s1::A

The results mirror that of the sub-L-system derivation using discrete time steps.

9.4 Music Generation

Interpretation of cell states is not limited to geometric constructs. Using an object-oriented approach allows the interpretation of cell states by methods other than a turtle interpretation. To provide different interpretations a collection of global modules are imported into the namespace of a particular system (the root system by default). These modules are directly interpreted as generative commands.

The musical commands use the concept of a state-based *player*, which is responsible for converting commands into actual music. The player maintains a state that includes the current pitch (note) and volume. The player converts incoming commands into MIDI[67] messages, enabling any MIDI compatible device to play music generated by the system. A small subset of the music generation command set is detailed below.

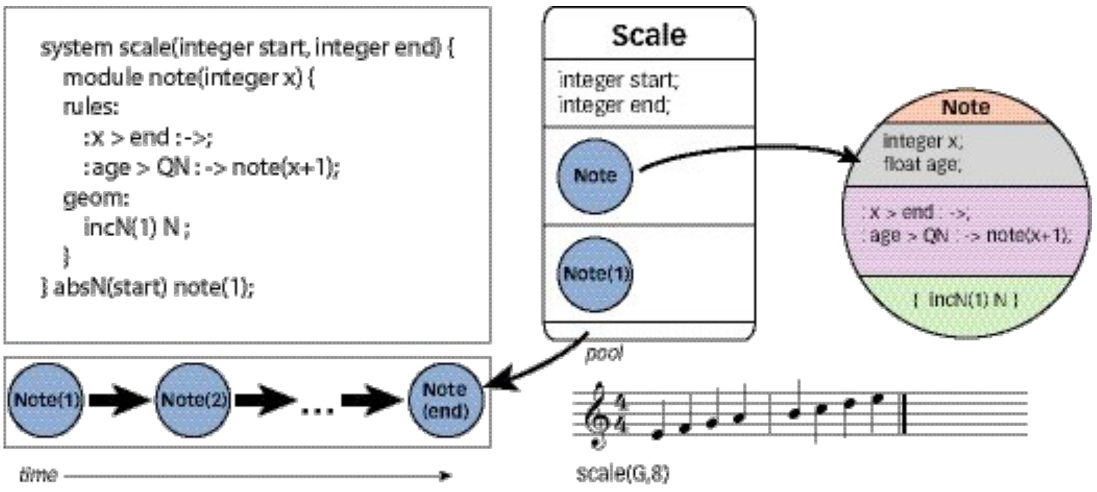
Module	Interpretation	Module	Interpretation
`absN(x)`	Set the current note to x, where x corresponds to the absolute pitch (the current key).	`velN(x)`	Set the current note on velocity to x. Velocity is converted to MIDI note-on velocity, in the range 0 to 127.
`incN(x)`	Change the current note by x semitones (x may be positive or negative)	`key(n)`	Set the key signature to the value specified by n.
`N`	Play the current note.	`QN`	A constant representing the duration of a quarter note.

Table 9–2: A selection of modules used to create music.

Figure 9-13 illustrates a simple system that plays a musical scale. The system scale takes two parameters that specify the starting and ending notes. The

[67] MIDI is a low-level serial communication protocol for musical instruments and musical controllers — see (Selfridge-Field 1997). Further details can be found in Chapter 11.

system's pool contains the developing notes that advance with quarter note timing. The `absN` module is used to set the player to the starting note. The `N` module actually plays the note, which is converted to MIDI note-on and note-off messages that correspond to the birth and death times of the module `note`. When `note` exceeds the value specified by the system parameter `end`, the pool becomes empty (recall that a rule's priority is implicit in its ordering). After the last note module has died and the system pool has emptied, the system continues to age, but no further development takes place in its pool.



```
system scale(integer start, integer end) {
  module note(integer x) {
  rules:
    :x > end :->;
    :age > QN : -> note(x+1);
  geom:
    incN(1) N ;
  }
} absN(start) note(1);
```

Figure 9-13: A note playing system, consisting of a single system, 'scale' which contains the module 'note', shown as a human-readable programming language (top left), a graphic representation as per the previous figures (top right); a symbolic developmental sequence (bottom left); and in music notation (bottom right). The system plays a scale from the supplied starting pitch for the given number of notes.

Musical applications are discussed in detail in Chapter 11.

9.5 Summary

The developmental system described here unifies a number of previous L-system models to the application domains of time-based development of aesthetic systems. Examples of such systems include generative animation and music synthesis. The temporal development, based on both discrete cellular changes and continuous development, successfully integrates these two modes of development, and permits complex temporal sequences not achievable using DOL-systems (Figure 9-14).

The hierarchical nature of the cellular developmental system allows management of complexity from the point of view of the user specifying a system model. Hierarchical ordering increases the control over structure at variety of levels, hence reducing the "brittleness"[68] of a flat grammar specification. This permits a more intuitive control over creation of modelled systems, either by the user (see section on user interface) or when using techniques such as aesthetic evolution (discussed in Chapter 10).

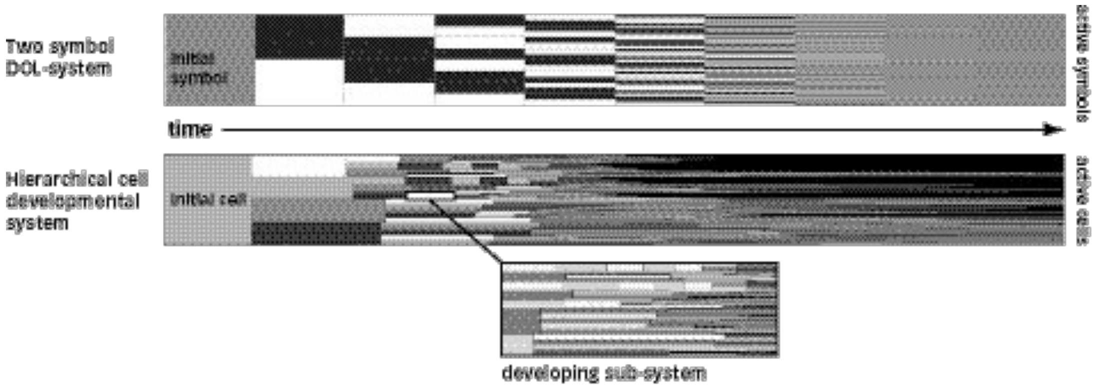


Figure 9-14: Time-state diagram contrasting the developmental differences between discrete L-systems and the cellular developmental system described in this chapter. The diagram shows the temporal development (from left to right) of each system. Shaded rectangles represent the symbols present (vertical axis) at any given time (horizontal axis). Both examples start from a single symbol or cell (the axiom). In the case of the DOL-System (top) each iteration is clearly synchronised and regular as the rewriting process proceeds in discrete time steps. In the case of the developmental cellular system, the sequence quickly becomes irregular and 'fractal' due to the individual developmental nature of cells. A single cell at the top level is shown expanded as a segment of a developing sub-system, illustrating the complexity that is contained by the hierarchy.

9.6 Implementation Details

The developmental algorithms for timed, parametric (and stochastic) L-systems, described in Section 8.2 of the previous chapter, cannot be used for the cellular developmental system described here, due to the indeterminacy of cell (module & system) ages when the cell is instantiated. The decision for a cell to split, die or produce additional cells will be based on state changes that may be determined by:

[68] That is, more robust to configuration changes — a change at one level in a hierarchy can have fewer side effects than for the equivalent description in a non-hierarchical system.

- external events or conditions generated from outside the system, such as interactive, real-time input from the human user or users of the system;
- context information of other cells that cannot be determined at the creation of the current cell;
- predicates involving state information influenced by incrementally calculated differential equations.

In addition, for applications such as music generation, real-time output and possibly synchronisation are required. For these reasons, an incremental algorithm is used to calculate cell states and an event-based framework used to instantiate and delete cells. Simulation of continuous state changes is achieved using a variety of numerical integration and interpolation techniques.

9.6.1 Media Synchronisation

As the system described in this chapter is used to generate temporal data expressed in different media types (sound, image, three-dimensional model, etc.), the system needs a way to synchronise this multimedia data to ensure correct timing. This is particularly important in the case of real-time applications, but is applicable to any temporal based interpretation of data generated.

Synchronisation defines the occurrence of multiple events at the same time, or within some small time period. In activities such as real-time animation or music synthesis, the generation of data needs to be synchronised to some global timing system, such as the computer's system clock or an external time code source. I will consider two types of temporal information:

- discrete, singular events with no duration that signify some change or point in the system as a whole;
- temporal segments of information with some duration in time.

Both types may be generated from temporal media *sources*. Synchronisation is generally concerned with the temporal ordering and output of events and segments from multiple sources in relation to some global timing information.

9.6.1.1 Related Work in Media Synchronisation

Recent surveys of media synchronisation describe a wide variety of schemes for synchronisation (Bertino & Ferrari 1998; Blakowski & Steinmetz 1996). The specification and organization of temporal segments and events can be broadly categorised into a number of different approaches. These include formal language-based models, where some form of organization is specified using a programming or scripting language (Ates et al. 1996). In graph-based methods, events and sources are represented as nodes with edges signifying constraints between them (Buchanan & Zellweger 1992). Similarly the declarative approach of Little and Ghafoor uses Petri nets which define temporal organization through places and transitions (Little & Ghafoor 1990). Such a system can be extended to maintain synchronisation over distributed networks (Qazi, Woo & Ghafoor 1993). While these systems provide considerable flexibility, they are difficult to construct from two perspectives. From a programming point-of-view, they require complex constraint solving methods (such a linear-programming) to be general. From the user perspective, it is difficult to manipulate graph representations as controllers for temporal sequences.

In contrast, *time-line* approaches place sources in temporal order, often using the concept of multiple *tracks* to support parallel event or segment synchronisation (e.g. the Apple *QuickTime* system (Towner 1999)). Such systems are popular in video and audio editing applications, since they lend themselves to intuitive and familiar graphic user interface paradigms, well established for such applications.

Many systems organise temporal elements using a *compositional* method, for example binary relationships between segments such as *before, starts, finishes, meets, overlaps,* and *during* (Allen 1983). In addition, hierarchical ordering or grouping relationships allow the collection of temporally unrelated sequences to form a unified sequence, which may be treated as a single entity in further operations, such as in the *Firefly* system of Buchanan and Zellweger (Buchanan & Zellweger 1993).

Ackermann describes an object-oriented framework to support multimedia application development, known as *MET++* (Ackermann 1996). MET++ has a versatile and extensible organization, particularly for the development and synchronisation of time-based media composed in a hierarchy (Ackermann 1993).

Like a number of systems, MET++ distinguishes between *intramedia* and *intermedia* synchronisation. Intramedia synchronisation handles continuity requirements within a single media source, and is primarily managed within the domain of the system generating that source. In contrast, intermedia synchronisation deals with the higher-level synchronisation of multiple sources and the temporal alignment of key events (such as starting and stopping of generative sequences) and segments (such as sounds or video segments). MET++ uses object-based event composition, whereby multiple sequences may be defined in a hierarchy, each with their own temporal structure, in much the same way as the combination of system and module cells form a hierarchical structure, allowing parallel and sequential temporal structures.

Relevant strategies for developmental simulation and synchronisation also come from the simulation community, where an important research goal has been the integration of discrete event based systems with those designed for continuous simulation of complex dynamical systems (Zeigler, Kim & Praehofer 2000). In many cases, this involves differential equation solvers with continuous states and continuous time formalisms being coupled with discrete state and discrete time simulations systems, such as automata systems. This is known in the simulation community as *multiformalism modelling,* where discrete and continuous formalisms are either combined together, or alternatively one formalism is embedded within another. It has been shown that a combination of each type of formalism is closed under coupling (Praehofer 1991).

The approach used here is an *object-oriented time synchronisation* approach, similar to that described by Ackermann (Ackermann 1996). However, the level of generality and flexibility provided by frameworks such as MET++ is unnecessary when dealing with the temporal development models required for the system described here. For example, the full scope of binary relationships is largely unnecessary, although there is no reason why a system could not include such additional features, particularly if it were to form part of a full multimedia system incorporating additional media elements.

9.6.1.2 *Temporal Development in the Cellular Model*

There are two distinct cases to be dealt with when simulating the development of cells:

- internal development of cell state where no new events such as creation, deletion or replacement of cells occur;
- changes to a system cell's pool, where cells are added, deleted or replaced.

The case of internal development can be further divided into *context dependent* and *context free* cases. In context free cases, the cell develops without dependence on surrounding cells. Cell state is determined only by changes to the internal state, which equates to solving differential or explicit equations to the point where a predicate changes value (similar to the forward prediction method detailed in Section 8.5.2.4).

For example, consider the rule of equation 9.4 on page 226. Here the rule must be integrated to the point where the predicate event occurs, i.e. where $x = X_{\max}$. This is equivalent to the root finding problem where we wish to find the first point $t_p, 0 > t_p \geq \tau$ such that the equation

$$g\big(t, x(t), x'(t)\big) = 0 \tag{9.8}$$

is satisfied at $t = t_p$. Shampine, Gladwell and Brankin describe a method to find the algebraic solution of this event location problem (Shampine, Gladwell & Brankin 1991) provided the solution can be defined as a polynomial in either x or x'. Such a technique can be integrated into a standard fourth or fifth-order Runge-Kutta numerical integration algorithm (Horn 1983).

In the context sensitive case, changes in cellular state depend on the states of neighbouring cells. Potentially, this dependence may cascade across all cells in the current pool (including system cells). In the general case, it becomes impossible to forward predict when a given cell's predicate state will change without actually simulating the entire system.

9.6.1.3 *Cellular Updates*

As discussed in the previous section, cellular updates can be classified as internal, involving only state changes, or external, involving modification of the system pool. As defined in Section 9.2.3 cellular rules are ordered sets of predicate-action sequences. Each predicate-action sequence is specified in order of evaluation in the cell's definition. Typically, predicate-action sequences specify state changes first followed by any cell actions. Once a cell action has been performed, no further predicate-action sequences are evaluated for that cell during the current time step.

The basic algorithm for evaluation of an individual cell's predicate-action sequences is shown below. This algorithm is executed at each time step to update the cell.

```
// Cell update algorithm
// This algorithm executes over a single time step
// All accessible variables are contained in a StackFrame
// We assume access to the module's pool via its parent system

for each predicate-action sequence $r_i$ {
   Boolean matched = FALSE;
   Boolean contextMatch = FALSE;
   ContextReference contextRef; // storage reference for context
   // activate (push) the local stack frame
   StackFrame.activate();
   if ($r_i$.hasContext()) {
      // check if the current context matches
      contextRef = system.pool.contextMatch($r_i$.context())
      if (contextRef.matched()) { // this rule has a context match
         // we need to add the context matched symbols before pro-
         // cessing the predicate
         StackFrame.add(contextRef.getStackFrame());
         contextMatch = TRUE;
      } else {
         StackFrame.deactivate();
         continue; // no context match so move on to next rule
   }
   // evaluate the predicate. We only get to here if
   // (i) there is no context to match, or
   // (ii) there is a context, which has matched.
   matched = evaluate($r_i$.predicate(),StackFrame).asBoolean();
   if (matched) {
      if ($r_i$.isAction()) {
         // this sequence is an action, requiring cell updates
```

```
            performActions($r_i$,StackFrame, system);
            // end the evaluation of sequences.
            StackFrame.deactivate()
            return;
        } else {
            // state update only
            evaluate($r_i$.stateCalculations(),StackFrame);
        }
    }
    // pop off any local variables
    StackFrame.deactivate()
}
```

Figure 9-15: Cellular update algorithm

Access to all variables is via an object known as a *stack frame.* Stack frames are hierarchical data structures (stack of ordered lists) created during compilation of cell specifications. Stack frames contain space for storage of variables (including temporary variables and intermediate results). They can be *activated* (pushed) and *deactivated* (popped) corresponding to changes in scope. Stack frames are similar to those used internally by compiled imperative programming languages to allocate space for local variables in function calls.

A successfully matched context may introduce new variables into the scope of the current predicate-action sequence being processed. This is because the publicly accessible state of context-matched cells may be used in the current cell's own state calculations or actions. A special object, `contextRef` is returned by the context matching checker `checkContext`. This `contextRef` object contains a local stack frame, which can be added to the current frame, giving any state calculations or cell actions access to the appropriate context-matched cell variables. Context matching is discussed in detail in the next section. Any local frame added to the `StackFrame` object will be removed upon call to the `deactivate()` method.

The function `evaluate` takes a list of statements (compiled as multiple parse trees) and evaluates them in the context of the supplied `StackFrame`. The result of the final statement is returned by `evaluate`. `evaluate` is used to check the predicate and perform any state calculations as required.

The function `performActions` takes the current predicate-action sequence, the `StackFrame`, the parent `system` and performs the necessary cell actions (described in Section 9.2.3.4) as dictated by the action component of the predicate-action sequence. Updating the system's pool is a two-step process — all updates occur in a buffered copy of the system pool. Once all cells in the pool have been processed (and potentially updated or changed), the buffered copy replaces the old pool. This method allows all cells to appear to be updated in parallel, with changes to the pool not taking place until the end of the current time step.

9.6.1.4 Cell Pools and Context Sensitivity

Cells develop in a *pool* — a data abstraction used for storing developing cells. The pool class hierarchy is implemented begins with an abstract base class with polymorphic functions to access, modify and context match cells (see Figure 9-16 below).

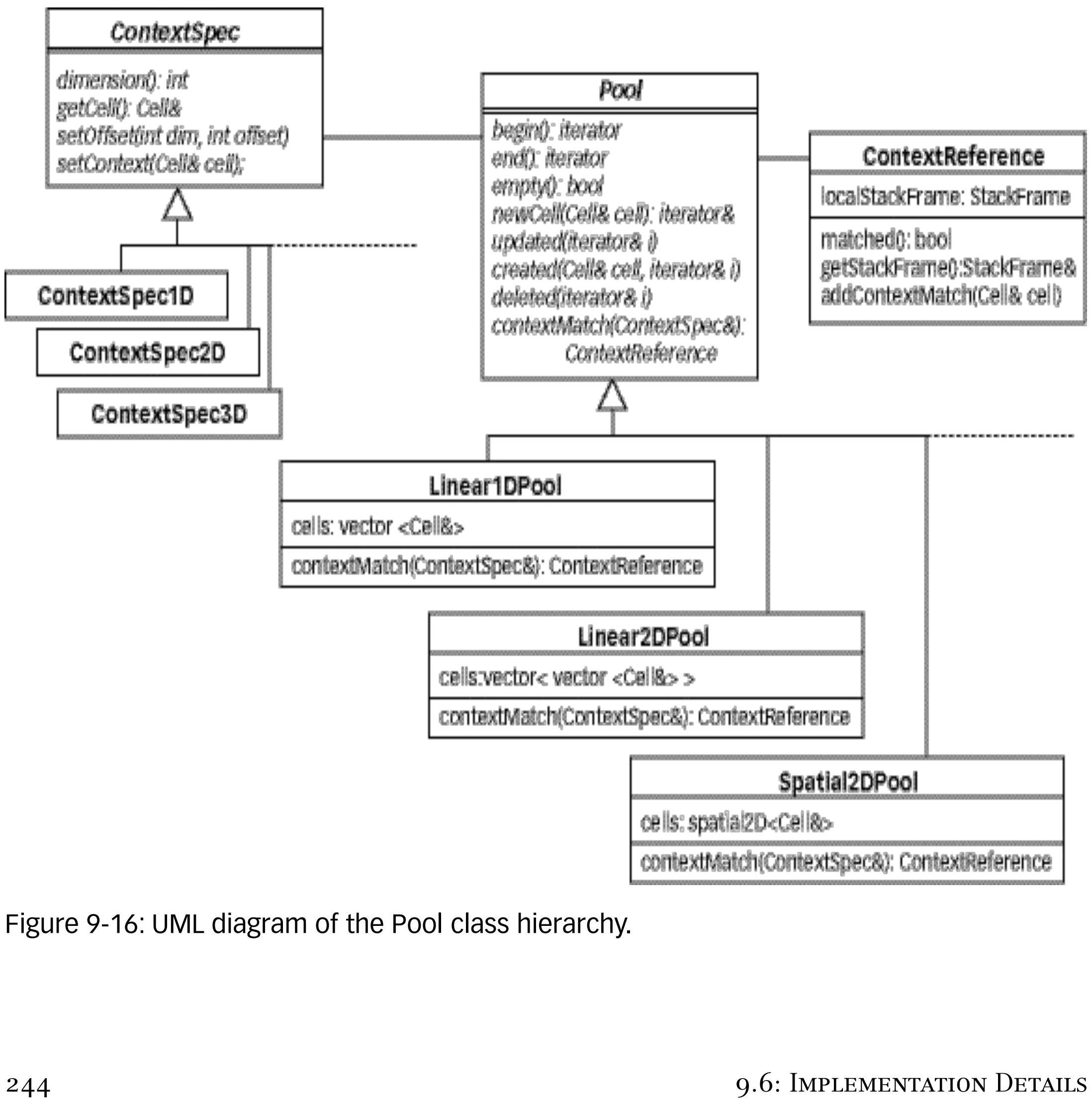

ContextSpec
dimension(): int
getCell(): Cell&
setOffset(int dim, int offset)
setContext(Cell& cell);
ContextSpec1D
ContextSpec2D
ContextSpec3D
Pool
begin(): iterator
end(): iterator
empty(): bool
newCell(Cell& cell): iterator&
updated(iterator& i)
created(Cell& cell, iterator& i)
deleted(iterator& i)
contextMatch(ContextSpec&): ContextReference
ContextReference
localStackFrame: StackFrame
matched(): bool
getStackFrame():StackFrame&
addContextMatch(Cell& cell)
Linear1DPool
cells: vector <Cell&>
contextMatch(ContextSpec&): ContextReference
Linear2DPool
cells:vector< vector <Cell&> >
contextMatch(ContextSpec&): ContextReference
Spatial2DPool
cells: spatial2D<Cell&>
contextMatch(ContextSpec&): ContextReference

Figure 9-16: UML diagram of the Pool class hierarchy.

The `pool` class forms an abstract interface through which cells can be updated, added and deleted. The `pool` class has an `iterator` class to permit polymorphic iteration over the actual data structure. A series of concrete classes provide the actual data structures to store cells. The classes `Linear1DPool` and `Linear2DPool` provide one- and two-dimensional arrays. Other, more specialised classes are also possible, such as the `Spatial2DPool` that can store cell spatial location and use Euclidean distance measures in context relations.

Two additional classes support n-dimensional context sensitivity. The `ContextSpec` abstract base class defines an interface for specifying context relations. Concrete classes are used to store actual context specifications based on dimension. A context specification, such as:

$$\big(A(x)\big)\big(C(z)\ me\ D(w)\big)\big(B(y)\big) \tag{9.9}$$

is a two-dimensional context, and so would be stored using a `ContextSpec2D` object. As part of the parsing of cell specifications, text specifications like the one above are converted to `ContextSpec` objects.

The `ContextReference` class is used to support handling of data following a context match. To test for a context match a `ContextSpec` object is passed to the pool object's `checkContext()` method. The dimension of the `ContextSpec` object must match that of the concrete pool subclass (this can be checked using the C++ run-time type identification system). It makes no sense to check for a two-dimensional context match in a one-dimensional pool, for example. Internally, the `checkContext()` method of the pool object checks for a context match and returns a `ContextReference` object. This object is returned regardless of the success or failure of the match.

As shown in Figure 9-15, the fist thing usually done to a returned `ContextReference` object is to check if it represents a successful match, via the object's `matched()` method, which returns `TRUE` if the context match was successful. The `ContextReference` object creates a local stack frame, which allows the pubic state of cells used in context relations to be incorporated into expressions used in either predicate matching or state calculations. As individual context cells are matched, they are passed to `ContextReference::addContextMatch()` method which extracts the cell's state information as it builds the local stack frame. So for example, the context relation given in (9.7) requires that the state variables *x, y, z, w* be available for use in the current predicate-action sequence, as they may be used as elements of predicate expressions and state calculations. All variables used from context relations are read-only and cannot be modified.

9.6.1.5 Simulation System Design

The basic building block in the object-oriented approach adopted for this system is a class representing a *time event* (class `TEvent`). This class forms the basis of a class hierarchy with subclasses inheriting the timing and synchronisation features of `TEvent` (refer Figure 9-17). The overall representation of time is handled by the `Time` class, which is capable of nanosecond accuracy in its representation of time[69], and provides various arithmetic and relations operators. Various utility classes allow the conversion to other time formats such as SMPTE time (hours, minutes, seconds, frames, sub-frames).

The `TEvent` class provides a number of methods to set and access time and offset values, calculate time positions and generate performance data over specified time intervals. The class hierarchy follows a compositional design pattern (Gamma 1995), whereby groups of `TEvents` can be treated as composites (class `TCompositeEvent`). This recursive structure allows collections of complex event sequences to be composed from collections of more primitive sequences, with all sequences complying with the `TEvent` interface.

Two composite sequence classes, `TSequence` and `TSynchro` are shown in the diagram. These classes provide specific temporal management policies, allowing a variety of temporal arrangements in compositions. For example, `TSequence` places multiple `TEvent` objects in a linear sequence and `TSynchro` synchronises groups of events, so that they all occur at the same time. A collection of support classes manage timing offsets and editing, in collaboration with the functionality of the `TEvent` base class.

[69] While the accuracy of representation is in nanoseconds, not all operating systems support such fine granularity in their system clock resolution, so the actual timing accuracy may not be possible at nanosecond time scales.

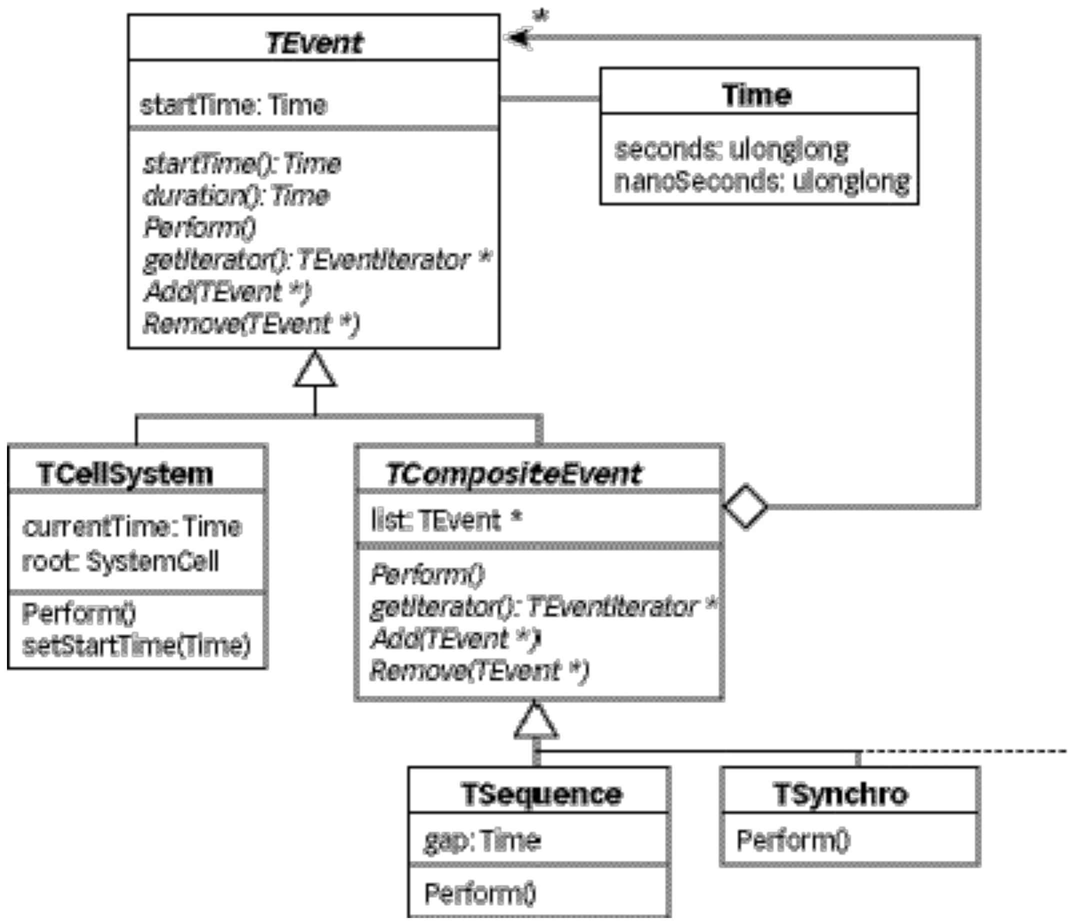


Figure 9-17: UML diagram of the timed event class hierarchy.

The `TCellSystem` class is the primary object used to control the development of a cellular system. This class contains references to the root system and stores the current local and global development time of the system it contains. All time-based development in client classes is achieved via the `TEvent` interface, allowing the integration of developing cellular systems with other time-based media elements, such as digital video or other generative systems.

9.6.1.6 Controlling Temporal Development

The function:

```
Perform(Time startTime, Time duration, Time realDuration)
```

is the primary method for temporal development. Before discussing this function in detail, the control mechanisms and synchronisation background will be explained.

As previously stated, synchronisation requires that temporal events occur at specific times. For example, in the case of PAL television animation generation a new frame must be generated at exactly every 25th of a second.

The basis of the time synchronisation and control system is managed by the `Conductor` class (see Figure 9-18), which provides basic mechanisms for controlling synchronisation and development of timed events. Conductor provides a number of standard transport functions, familiar to users of video tape recorders, for example (*stop, rewind, play, pause, step*).

The conductor is driven by a low level timing system, encapsulated by the `Ticker` class. A ticker is a low-level interface to various forms of timing information, including real-time information (driven by system interrupts) or timing information from remote timing sources. A specific `Ticker` object is attached to the conductor, repeatedly calling the conductor's `Notify` method at each clock "tick". The conductor specifies the time interval, δ, between calls to the `Notify` method. This interval represents the finest quanta of timing granularity that the system can achieve. This does not mean events cannot happen inside this interval, just that synchronisation events resolve only to this interval in terms of accuracy.

It is the responsibility of the ticker to maintain correct interval durations and translate any internal or external timing information as necessary (for example an external timing source may interrupt or resolve to a different interval time (δ) than what the conductor requires, so some internal translation will be necessary).

The interval, δ, does not necessarily represent real or actual time. In a remotely driven sync situation, incoming timing information may vary significantly from real-time, however the ticker simply maintains a representation of duration according to either an internal or an external reference.

`Ticker` is designed to work as a separate thread (light-weight process), running independently of the `Conductor`, or alternatively in "blocking" mode where it may be part of some larger event loop.

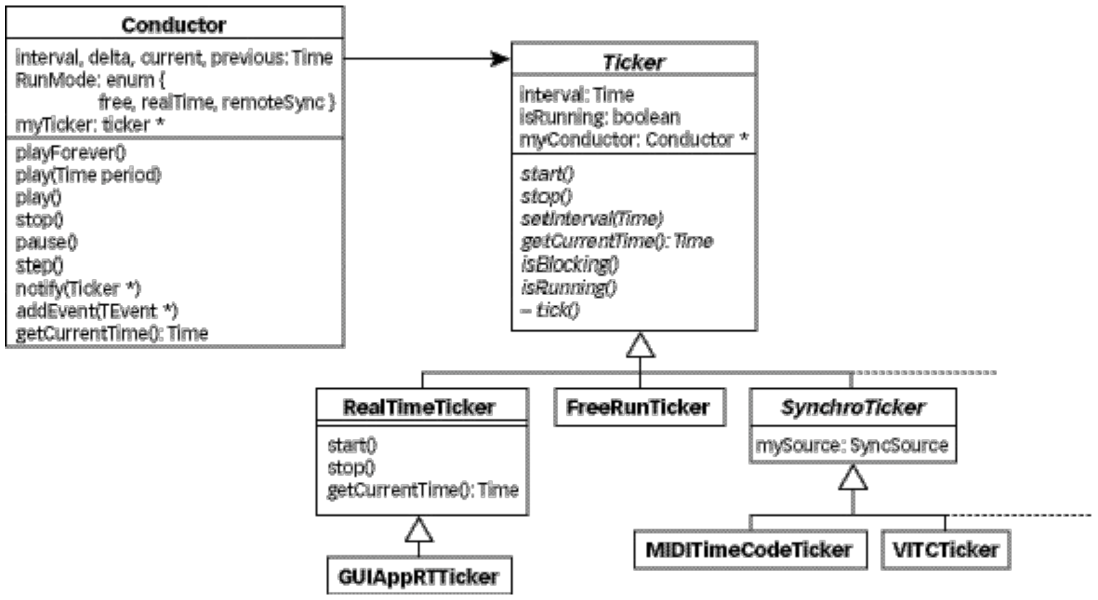


Figure 9-18: UML diagram of time control and synchronisation classes.

The sub-classes of `Ticker` implement a variety of timing strategies. `RealTimeTicker` is a threaded, interrupt driven class that uses system timing information to provide `Notify` calls in real time. A typical interval in this case would be 40ms for generating real-time animations. `GUIAppRTTicker`, a sub-class of `RealTimeTicker` is a special class designed to integrate with GUI event loops, using threads to maintain user-interface responsiveness, but also ensuring correct synchronisation with frame-buffer redraw events scheduled to coincide with vertical redraws on a graphics display, such as frame buffer swaps in double-buffered displays (Angel 2003).

`FreeRunTicker` is used when maximum performance is required. It simply calls the `Conductor`'s `Notify` function as fast as possible, providing maximum performance. `Notify` will block in this mode, ensuring that there are no `Notify` calls pending until computation for the specified interval has been completed. `FreeRunTicker` is useful in situations where maximum throughput is required, for example in the generation of off-line animated sequences, where models are output to a software renderer that does not operate in real time.

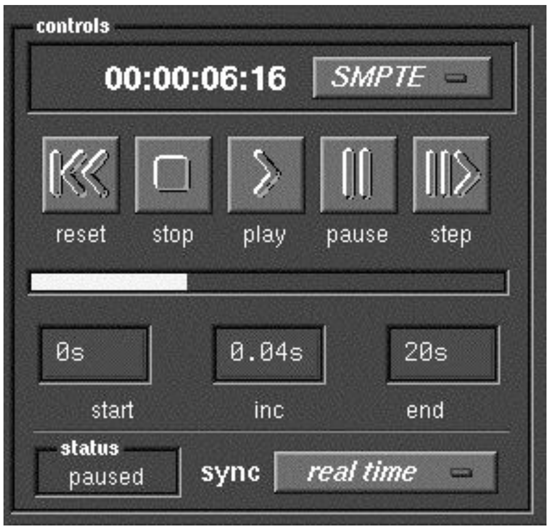


Figure 9-19: Conductor GUI, showing transport, timing and synchronisation controls. The user may set start, end and update (increment) times via text boxes. The time bar (centre) shows progress from start to end. Current time is displayed at the top (a variety of formats can be selected). Synchronisation can be selected from a number of available sources, and even changed while the system is running. A status box shows the current status, including synchronisation locking if the conductor is locked to an external sync source.

The calling sequence of `Ticker` and its sub-classes is shown below in Figure 9-20.

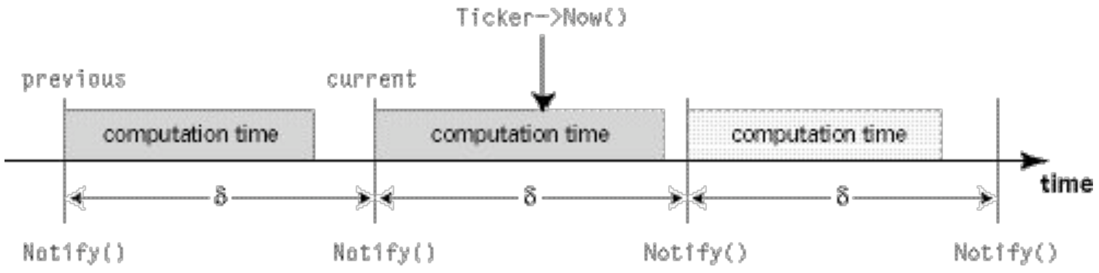


Figure 9-20: Calls of the `Ticker::Notify()` function at regular intervals. Computation, controlled by the `Conductor`, is performed after receiving each `Notify` event (using the `Perform()` function). Further calls from `Notify` are blocked until the computation for the current time-step is complete.

As alluded to in the beginning of this section, the `Perform` function processes the necessary computation for the specified interval. This involves traversing the composite `TEvent` hierarchy, and developing the state of the system from the current time step until the next. This may involve further sub-divisions of time, for example when solving the differential equations of a cell's state using an adaptive integration technique, such as the Adams-Moulton method (Kreyszig 1999, Section 19.2).

Execution of the `Perform` function is further divided into pre-processing, inner-processing and post-processing stages. This provides a way to prioritise processing, for example buffer allocation in the pre-processing stage, actual generation in the inner stage and buffer flushing in the post-processing stage.

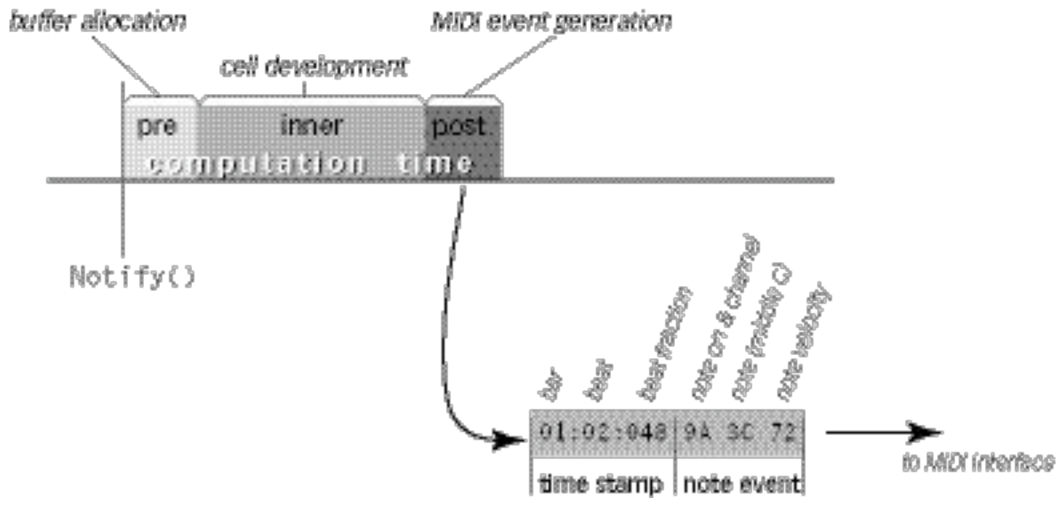


Figure 9-21: The three stages of processing within a single interval. In the example shown here, the pre-processing stage allocates buffers, the inner stage controls cell development, resulting in the generation of MIDI events that are placed in the buffer. The post stage sends the MIDI events to the appropriate MIDI hardware. The format of an individual MIDI message is also shown — containing time stamp and note event information.

For example, when generating musical data via the MIDI protocol, a series of MIDI events must be generated and placed into a buffer. Each MIDI event consists of two components — a time stamp (in bars, beats and fractions of a beat or SMPTE time code) and event information (Hewlett & Selfridge-Field 1997), encoded as byte sequences. A buffer containing these low-level MIDI events is then passed to the MIDI interface system (MIDI interface hardware with software interface), which will output the MIDI events at the appropriate time. At each interval, the buffer requires all the events needed for the current interval.

9.6.1.7 Calculations that Exceed the Time Interval

`Perform` performs the calculations necessary to advance the system to the next time step. The diagram in Figure 9-20 implies that any necessary computation can be performed within the time interval between `Notify` calls from the current `Ticker`. However, it is possible that a cellular developmental process cannot be bounded by a given time interval, and that computation time will exceed such an interval. In this case, the system tries to adapt by following a number of strategies:

- System pool growth is monitored and the pool's owner system development is terminated if the pool exceeds a certain size. This stops any system from unbounded exponential growth. E.g. the simple program:

```
system grow {
   module a {
      : age > timestep : -> a a;
   }
} a
```

System grow's pool doubles in size at each time step. The termination size can be user specified. If no value is specified then the value is inherited from the parent system (the root system has a default size, which can be modified).

- The Conductor class operates under an adaptive interval scheme (detailed below).

At each time interval, the total computation time (Figure 9-20) for that time interval is calculated. The difference between computation and interval time is also calculated, and stored in the parameter delta. The value of delta is monitored and the Conductor class tries to adapt based on this value and the desired time interval.

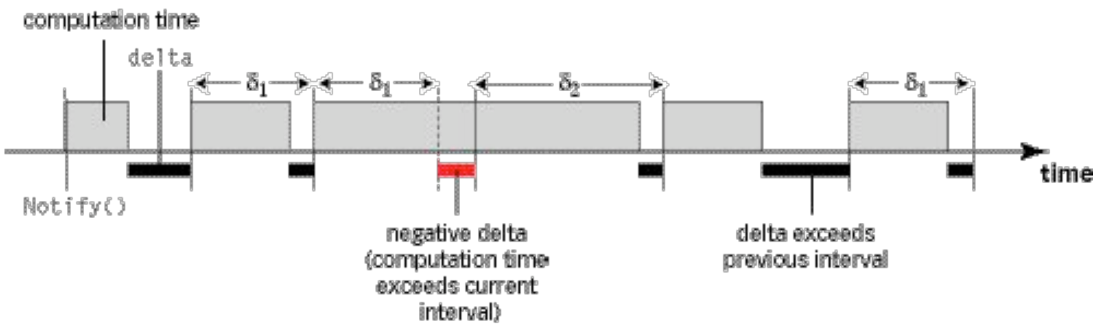

computation time
delta
δ1
δ1
δ2
δ1
time
Notify()
negative delta (computation time exceeds current interval)
delta exceeds previous interval

Figure 9-22: Recalculation of intervals if computation time exceeds Ticker intervals.

If the delta parameter becomes negative (i.e. computation time for the current time step has exceeded the real-time required for its calculation), the ticker is notified to increase the interval δ by a factor linearly dependent on delta. This strategy results in a smaller number of updates than desired, while attempting to maintain real time performance. If the value of delta exceeds the original desired interval time δ will revert to its previous value. Larger values of δ may eliminate overheads such as building complex geometric models and allow numerical integration schemes to work more efficiently. Limits are placed on the maximum and minimum values of δ to avoid exponential or unrealistic increments in pathological cases (a system that updates once every ten minutes is not particularly useful in a real-time application).

The above method works as a simple optimisation scheme that tries to balance real time performance with target update intervals and synchronisation accuracy. However, it is always possible to create developmental systems that are too complex to be computed in real-time. In this case, the system cannot maintain real-time performance and deliver an accurate result, so the value of δ remains at its maximum and updates continue as fast as possible with current resources (essentially mimicking the behaviour of the `FreeRunTicker`). In such a situation, any accurate synchronisation with an external or real time source will be lost. Other media streaming systems that deliver "pre-computed" media (such as video) usually adopt the strategy of skipping or dropping frames in order to maintain synchronisation. In the case of developmental system development, this is not possible because the state of the system is dependent on its state at the previous time step.

9.6.1.8 Results

Provided the complexity of the system permits realistic computational times for the chosen interval, the method described in this section is capable of maintaining real-time performance and accurate synchronisation with an external time source. This system has been used for real-time musical performance, where even on a modest computer (250Mhz SGI Indigo) the system can maintain real-time output of complex musical data generated using the developmental system described in this chapter.

9.6.2 User Interface

The system described here is intended for use in the creation of artistic works, realised as animations, interactive simulations or musical compositions. Whatever technical capacity an artistic software system may have, the sensuous interaction between human, machine, and software is surely a limiting factor for anything that system produces. This section details the procedural qualities of the system in terms of how a user might interact with the system in the process of developing these works.

Cells may be specified using the programming language described in Section 9.3. Interpretation of modules is dynamically selected by the inclusion of a series of modules that determine how the informational properties are interpreted as (for example) geometry, sound, text, and so on.

Writing cell systems using the built-in programming language can be tedious and difficult for a non-expert. As an attempt to alleviate this problem, two strategies have been developed:

- using a graphical user interface (GUI) that allows graphical manipulation of the cellular hierarchy and editing of the rules of individual modules;
- using an aesthetic evolution technique whereby the user navigates the search space of possibilities by selection and mutation (described in Chapter 10).

Both these strategies borrow from the biological analogy in my definition of generative art (Section 3.6) — that of genotype and phenotype. With the genotype being the developmental system specification: a hierarchical specification of systems and modules. The phenotype is the animated form or musical composition generated by the developing genotype.

9.6.2.1 The "gene splicer" editor

In the case of the GUI–based system (hereafter referred to as *gene splicer*), the software provides a graphic representation of genotype, in the form of a tree, the nodes of the tree representing cells (refer to Figure 9-23 and Figure 9-24). A tree structure is used, as it is capable of representing the hierarchical definition of cells, with system cells at parent nodes and master cells at the leaves. The root system is always at the base of the tree.

In interaction with the system, one can select various branches of the tree and perform simple editing operations, such as removal, insertion and replacement — a kind of "gene splicing" after which the program is named.

Double-clicking on a box representing a cell opens up a window that permits editing of that individual cell (Figure 9-25). A library area holds collections of systems from which the user may assemble their own new collections.

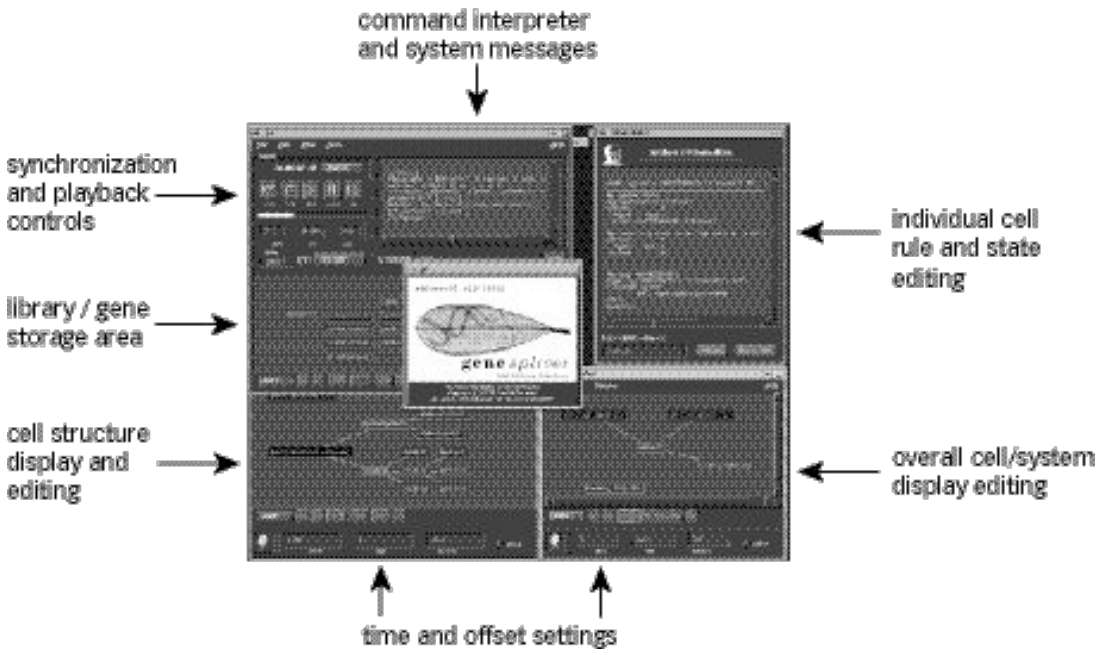

command interpreter and system messages
synchronization and playback controls
library / gene storage area
cell structure display and editing
individual cell rule and state editing
overall cell/system display editing
time and offset settings
gene splicer

Figure 9-23: Graphical User Interface for the *gene splicer* software, showing the key user interface elements accessible to the user.

Time control uses the common "VCR analogy" providing controls to play, pause, stop and rewind. No "fast forward" control is currently provided. The rewind function resets the current root system to its initial state, including its pool (and hence all sub-systems, their pools and modules).

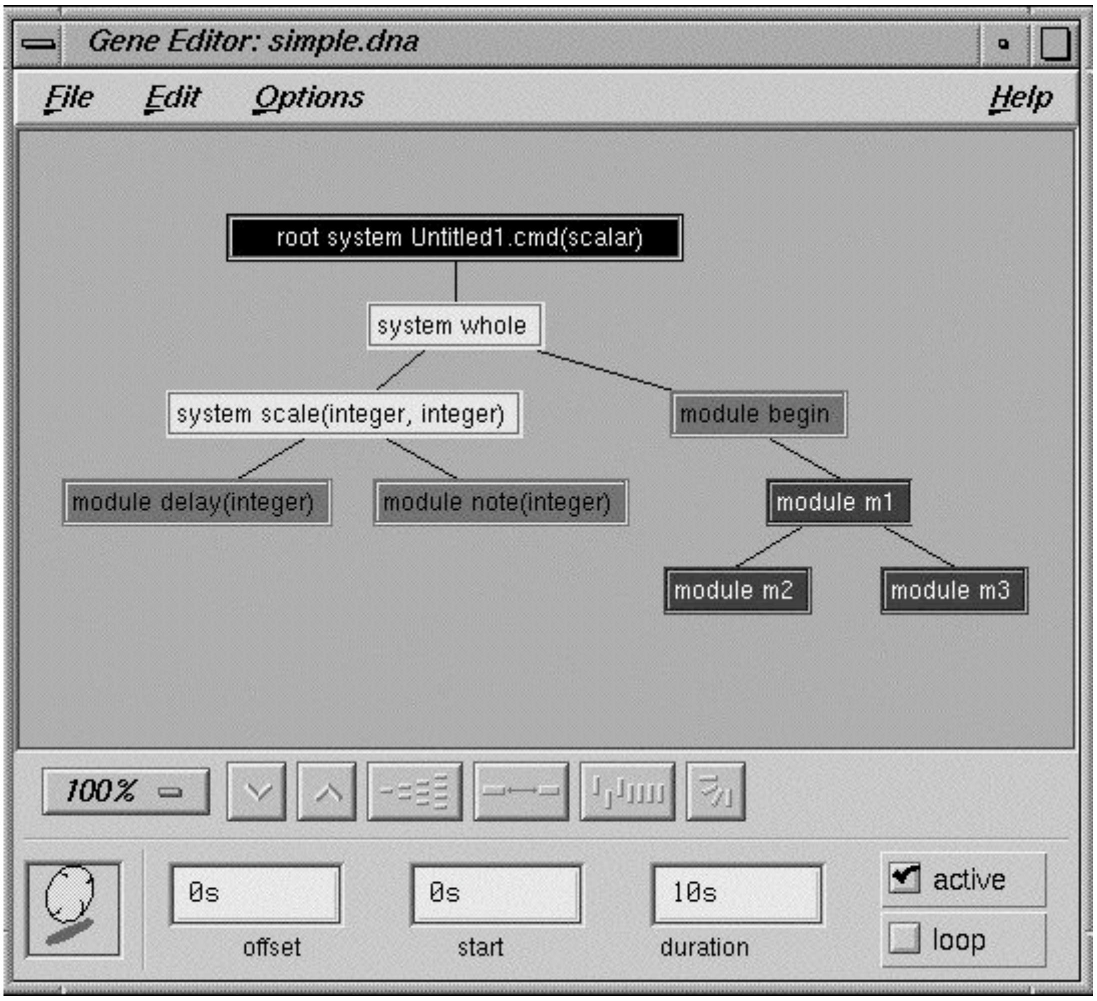


Figure 9-24: Editing section of *gene splicer*, showing the hierarchical structure of a collection of cells.

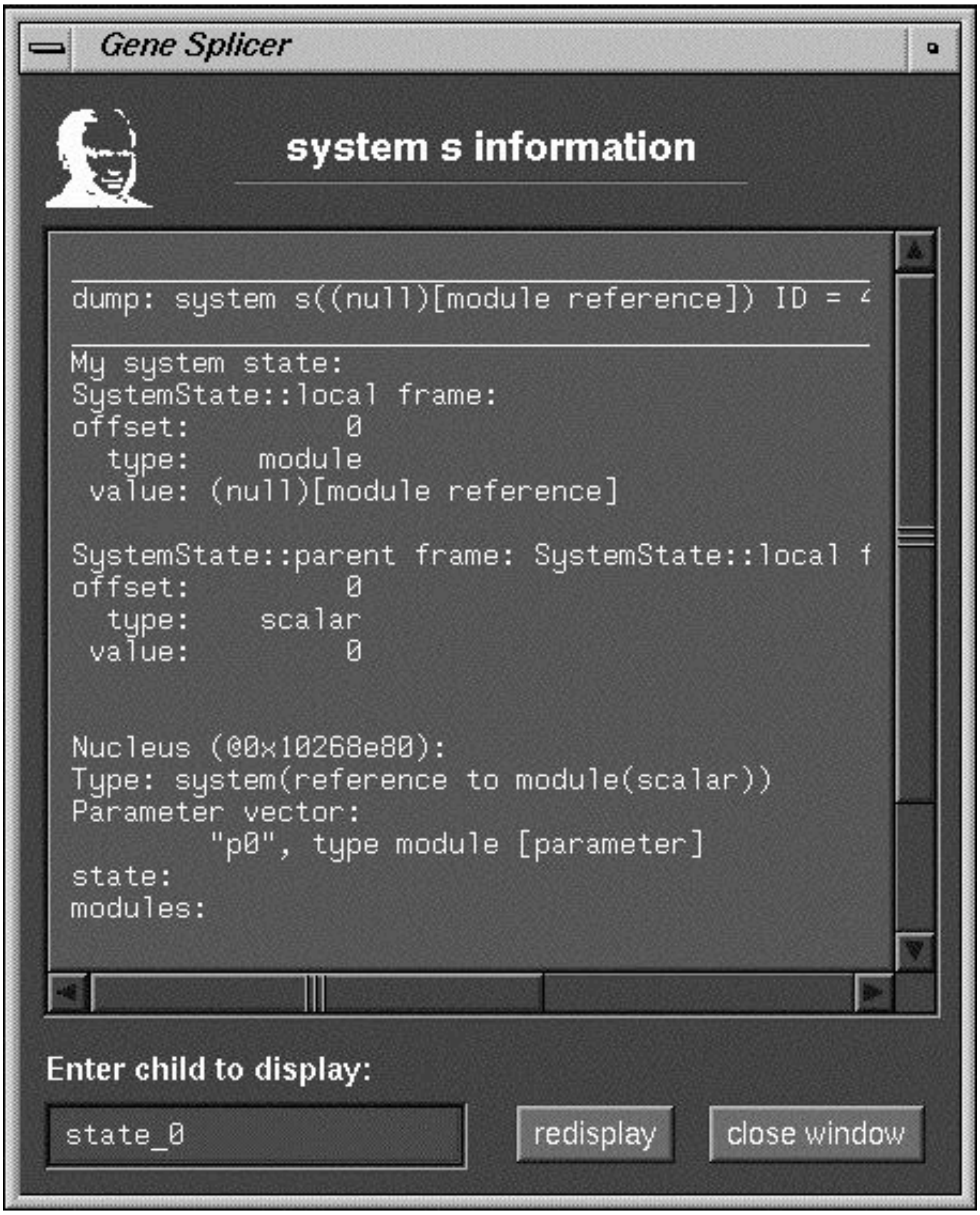
Three basic time synchronisation modes can be selected:

i. real time playback, synchronised to the computer's internal clock;

ii. real time playback, synchronised to an external time signal (MIDI time code);

iii. no synchronisation — development proceeds at maximum performance limited by CPU capabilities.

Details on the internal implementation of these modes were discussed in Section 9.6.1. Multiple systems may be running simultaneously. Each system has its own window with controls for timing offsets allowing certain events to begin at specified times. This is particularly useful when synchronising with external events via midi time code, for example. All time information may be specified in SMPTE format[70] (hours:minutes:seconds:frames:sub-frames) see (CipherDigital 1987).



```
dump: system s((null)[module reference]) ID = 4

My system state:
SystemState::local frame:
offset:         0
  type:    module
 value: (null)[module reference]

SystemState::parent frame: SystemState::local f
offset:         0
  type:    scalar
 value:         0


Nucleus (@0x10268e80):
Type: system(reference to module(scalar))
Parameter vector:
        "p0", type module [parameter]
state:
modules:
```

Figure 9-25: Text display/editing of individual cells.

[70] American National Standard for television — time and control code ANSI/SMPTE 12M-1986 (Society of Motion Picture and Television Engineers).

9.7 Conclusions

The developmental system presented in this chapter should be seen as a progressive development that integrates a number of advancements in L-systems research under a single model. These advancements include parametric and sub-L-systems (Hanan 1992), differential L-systems (dL-systems) (Prusinkiewicz, Hammel & Mjolsness 1993), timed L-systems (tL-systems) (Prusinkiewicz & Lindenmayer 1990) in addition to context sensitivity. The novel context relations allow the simulation of other developmental techniques such as cellular automata and reaction-diffusion systems.

In addition to this integration, a method for real-time synchronisation has also been described. Combining this with a flexible architecture for media generation results in a generative system capable of producing music performance data or 3D geometry in real-time. The use of a graphical user interface and the ability to splice systems and modules between different compositions gives the non-expert composer or artist the ability to create complex models in an intuitive way, without having to learn a complex programming language. The system described has been used to generate a number of successful compositions and animations, many of which are described in this thesis.

10 Interactive Evolution[71]

Any “novelty,” … will be tested before all else for its compatibility with the whole of the system already bound by the innumerable controls commanding the execution of the organism’s projective purpose. Hence the only acceptable mutations are those which, at the very least, do not lessen the coherence of the teleonomic apparatus, but rather, further strengthen it in its already assumed orientation or (probably more rarely) open the way to new possibilities.

—*Jacques Monod, “Chance and Necessity” (Monod 1971, page 119)*

Aesthetic evolution of L-systems provides a powerful method for creating complex computer graphics and animations. This chapter describes an interactive modelling system for computer graphics in which the user is able to evolve grammatical rules and surface equations. Starting from any initial timed, parametric D0L-system grammar, evolution proceeds via repeated random mutation and user selection. Sub-classes of the mutation process depend on the context of the current symbol or rule being mutated and include mutation of: parametric equations and expressions, development functions, rules, and productions. As the grammar allows importation of parametric surfaces, these surfaces can be mutated and selected as well. The mutated rules are then interpreted to create a three-dimensional, time-dependent model composed of parametric and polygonal geometry. L-system evolution allows a novice user, with minimal knowledge of L-systems, to create complex, “life-like” images and animations that would be difficult and far more time-consuming to achieve by writing rules and equations explicitly.

[71] This chapter is based on material first published in (McCormack 1993).

10.1 Introduction

Modern computer graphics and computer aided design (CAD) systems allow for the creation of three-dimensional geometric models with a high degree of user interaction. Such systems provide a reasonable paradigm for modelling the geometric objects made by humans. Many organic and natural objects, however, have a great deal of complexity that proves difficult or even impossible to model with surface or CSG based modelling systems. Moreover, many natural objects are *statistically self-similar,* that is they appear approximately the same but no two of the same species are identical in their proportions.

As has been shown in previous chapters, L-systems have a demonstrated ability to model natural objects, particularly branching structures, botanical and cellular models. However, despite the flexibility and potential of L-systems for organic modelling, (and procedural models in general), they are difficult for the non-expert to use and control. To create specific models requires much experimentation and analysis of the object to be modelled. Even an experienced user can only create models that are understood and designed based on their knowledge of how the system works. This may be a limitation if the designer seeks to explore an open-ended design space (as discussed in Chapter 4). The method presented here gives the artist or designer a way of exploring the "phase space" of possibilities offered by L-system models.

10.1.1 L-systems

The type of L-system primarily used in this chapter is *timed*, *parametric 0L-systems,* both deterministic and non-deterministic (see Section 8.2 for definitions). Evolution of context sensitive L-systems has not been attempted at this stage.

10.1.2 Evolution

Natural evolution, first proposed by Charles Darwin (Darwin 1859) and Alfred Russel Wallace (Wallace 1855) provides a theory for the development of species. Through the process of natural selection, a rich and complex diversity of organic life has evolved. The *genetic algorithm*, as proposed by Holland (Holland 1992) follows this evolutionary paradigm and provides a method for searching very large spaces for an optimal solution. Genetic algorithms have been used to solve a number of problems in design, optimisation and fitness (Goldberg 1989; Grenfenstette 1985, 1987; Rawlins 1991).

In his book, *The Blind Watchmaker*, Richard Dawkins demonstrated simulated evolution by evolving *biomorphs* — simple two-dimensional structures resembling organic creatures created from minimal sets of genetic parameters (Dawkins 1986). The survival of each generation of biomorphs is selected by the user who evolves features according to their personal selection.

Other applications of the genetic algorithm to image and object generation include Todd and Latham who have evolved computer sculptures using CSG techniques (Todd, S. & Latham 1991, 1992). Sims has evolved procedural models to create branching structures, textures, parametric surfaces and dynamic systems (Sims 1991b, 1991a, 1993).

10.1.3 Genetic Terminology

Genetic terminology is also applied in the case of these simulations. The *genotype* is the genetic information that contains the codes for the creation of the individual. In organic life, this is usually the DNA of the organism. In the case of simulations, the genotype can be a string of digits or parameters. In the case of L-system grammars, it is the productions, axiom, symbols and their associated parameters.

The individual, object or system created from the genotype is known as the *phenotype*. The process of *expression* (genome to phoneme) usually generates the complexity from the relative simplicity of the genotype.

Fitness of phenotypes is determined by *selection*. In a real world environment, the fitness of an organism determines its ability to pass on its genes from generation to generation — mating for sexual species, and survival. In the real world, many factors influence an organism's fitness to the task of survival. In simulated genetic systems, the fitness either can be determined automatically, by a defined *fitness function,* or, as is the case with this work, selected implicitly by the user.

10.1.4 Overview of the Aesthetic Evolution Process

In a natural situation, genetic variation is achieved through two key processes — *mutation* of the parent genotype and *crossing over* of two parental genotypes in the case of sexual species. In the system described here, "child" genotypes are created by *mutating* a single parent genotype. Such mutations cause different phenotypes to result. Changes may affect structure, size, topology and growth. Through a repeated process of selection by the user and mutation by the computer, aesthetic characteristics of the resultant forms can be optimised (the genetic algorithm can be thought of as an optimisation process). Figure 10-1 illustrates this process.

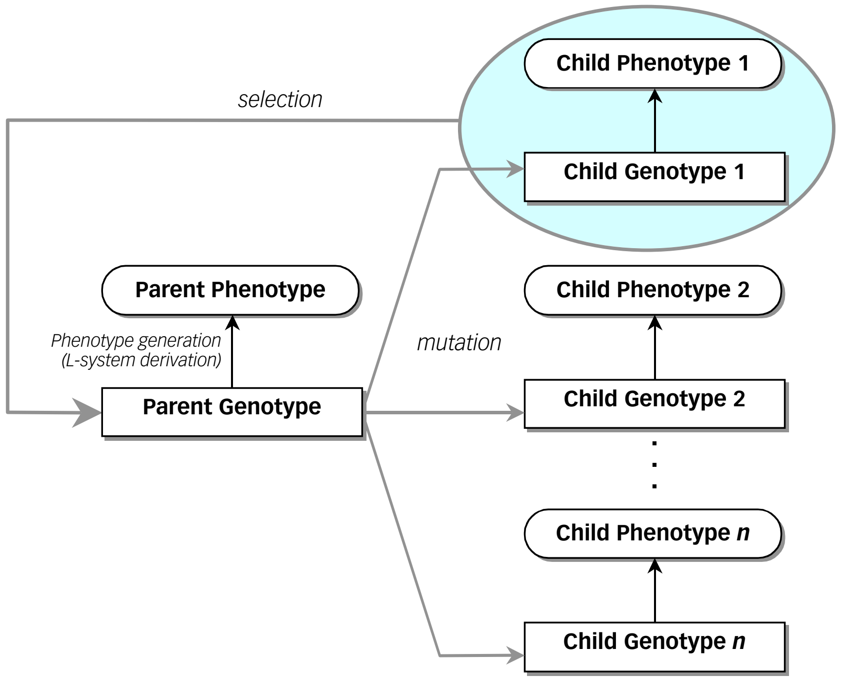


Figure 10-1: The mutation and selection process.

Typically, the user of the system begins with an existing L-system that they wish to evolve. This *germinal genotype* becomes the first parent from which offspring are mutated. A library of existing L-systems is available for the user to select from to begin the evolutionary process. It is also possible to evolve "from scratch", i.e. with only implicitly defined identity productions in the germinal genotype.

Following the evolutionary process, the user has the option of saving the mutated genotype in the library. Alternatively, the phenotype (three-dimensional geometric data, shading and lighting information) can be exported to software or hardware based rendering systems for further visualisation and processing. This process is outlined in Figure 10-2 below.

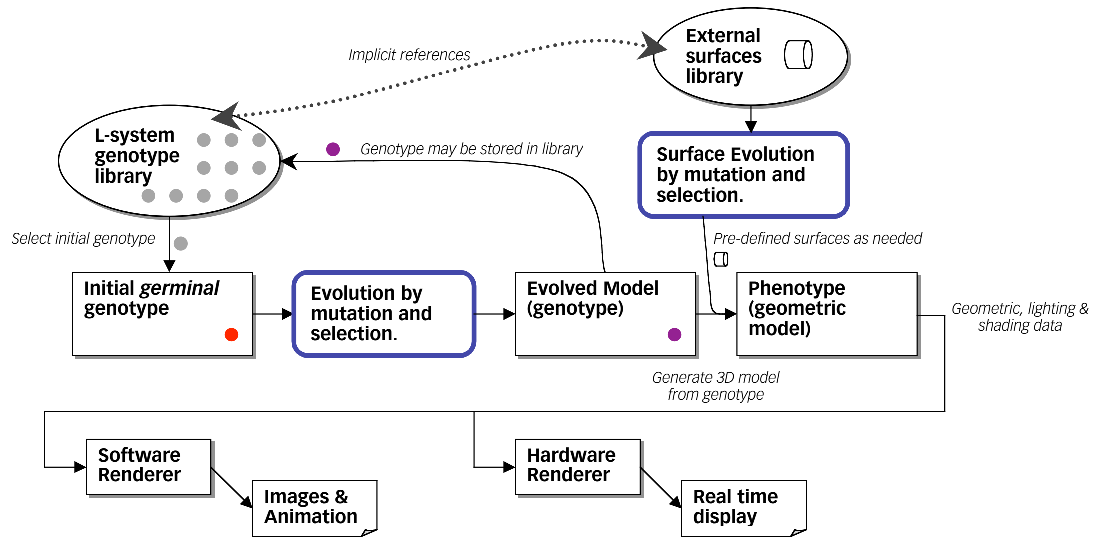

Implicit references
External surfaces library
L-system genotype library
Genotype may be stored in library
Surface Evolution by mutation and selection.
Select initial genotype
Pre-defined surfaces as needed
Initial *germinal* genotype
Evolution by mutation and selection.
Evolved Model (genotype)
Phenotype (geometric model)
Geometric, lighting & shading data
Generate 3D model from genotype
Software Renderer
Images & Animation
Hardware Renderer
Real time display

Figure 10-2: The initial genome is selected from an existing library to undergo the evolutionary processes described in this chapter. Following evolution the genotype can be (optionally) saved into the library or output to rendering systems for visualisation.

The next section looks at the syntax and structure of L-systems used. Section 10.3 examines the mutation process, Section 10.4 the interactive evolution process, and Section 10.4.1 discusses the implementation. Section 10.5 presents a summary of results and Section 10.6 possibilities for further work. Finally, Section 10.7 provides an epilogue on the original research presented in this chapter.

10.2 L-Systems Implementation

This chapter describes the evolutionary components of the modelling system developed for creating three-dimensional animated models using timed, parametric context-free L-systems (tp0L-systems). The foundation of the system is a parser/generator, which takes a set of rules and parameters and creates a time dependent geometric model as its output. The state of the system can be sampled at any chosen interval and a geometric model produced. For animations, this is typically 25 times per second. The system permits timed, parametric deterministic (D0L-) and non-deterministic stochastic (0L-) systems, both of which are context-free. Currently, context sensitive 1L and 2L-systems are not implemented as part of the evolutionary components of the program. Many of the results obtained with 1L- and 2L-systems can be achieved with parametric 0L-systems (Prusinkiewicz, Lindenmayer & Hanan 1988).

10.3 Mutation of L-system Rules

In order for the structure and form of an L-system generated model to change, its productions and module parameters must be changed. Small changes in genotype are usually preferred so as not to completely alter the form on which it is based. However, for radical change larger amounts of mutation are required.

There are three principal components of an L-system grammar to which mutation can be applied:

- mutation of rules (productions) and successor sequences (Section 10.3.1);
- mutation of parameters and parametric expressions (Section 10.3.2);
- mutation of development functions and symbol ages (Section 10.3.3).

In addition, pre-defined surfaces, included as part of the turtle interpretation may themselves be mutated, using, for example, techniques described by Sims (Sims 1991b). The connection between surface mutation and L-system mutation is illustrated in Figure 10-2. The two processes proceed independently of each other, coming together at the generation of the geometric model.

10.3.1 Rule Mutation

For each production, symbols and successors have the possibility of being mutated. For the moment, we assume a D0L-system $G = \langle V, \omega, P \rangle$ as defined in Section 5.3.2.1. Mutation of a single production can be represented $p_n \Rightarrow p_n^*$, where $p_n \in P$ is the original production and p_n^* the mutated version. The probability of each type of mutation is specified separately. The types of mutations possible are listed below:

A.1) A symbol from a production successor may be removed; e.g.: $p_n = \{a \to ab\} \Rightarrow p_n^* = \{a \to b\}$

A.2) A symbol α, where $\alpha \in \chi_{p_n}$ may be added to a production successor;

e.g.: $a \to ab \Rightarrow a \to abb$, where α is b.

A.3) A symbol α, where $\alpha \in \chi_{p_n}$ may change to another, different symbol;

e.g.: $a \to ab \Rightarrow a \to bb$, where a has changed to b.

The above three rules work on a previously defined set of symbols, typically a subset of the L-systems alphabet, $\psi : \psi \subseteq V$. This subset can be specified by the user. In addition:

A.4) A new symbol, α, where $\alpha \in \psi$ may be added to the production;

e.g.: $a \to ab \Rightarrow a \to abc$, where α is c in this example.

In addition, it is sometimes necessary to disallow mutation of certain symbols whose purpose is some kind of control (such as output resolution), or to limit a search space. Special symbols, such as the brackets ('[' and ']') representing turtle state *push* and *pop* operations, need to be matched to prevent a stack overflow or underflow. Here there are two options:

- ignore stack underflows and kill any genotypes that reach an upper limit of stack size when the phenotype is generated;
- execute a bracket *balancing* operation following the final mutation of a given production. That is for any given production ensure the number of "[" and "]" symbols is equal. This can be achieved by adding or deleting bracket symbols as necessary. This is the preferred option.

10.3.1.1 Production Set Mutation

In addition to individual symbol changes within successors, productions may be created and deleted. Here we assume a stochastic 0L-system, $G_\pi = \langle V, P, \omega, \pi \rangle$ as defined in Section 5.3.4.1.

B.1) A stochastic production p_n with probability $\pi(p_n)$ may split into two stochastic productions, p_n' and p_n^*, i.e. $p_n \Rightarrow \{p_n', p_n'^*\}$, where $\pi(p_n') + \pi(p_n'^*) = \pi(p_n)$.

For example: $a \overset{0.8}{\rightarrow} ab \Rightarrow \left\{ a \overset{0.3}{\rightarrow} ab, a \overset{0.5}{\rightarrow} bb \right\}$

The successor of p'_n is the same as that of p_n. The successor of p'^*_n is a mutated version of p'_n, created using the mutations *A.1–A.4* as specified in the section above.

B.2) A new production can be created. Essentially, this is for identity productions of the form $a \rightarrow a$, which are implicitly assumed for all members of V with no explicit productions. If all members of V have non-identity productions then this mutation can't take place. If an identity production does exist then the successor is assigned to be a randomly selected element from the set ψ (defined in the previous section). Note that this may result in the identity production if the predecessor is an element of ψ. This rule is superfluous to a degree, since if we assume the existence of identity productions they could be subject to the mutations *A.1–A.4* as specified in the previous section. It is introduced as a convenience due to the internal representation of productions in the system.

B.3) An existing production may be deleted. If the production is stochastic then the productions with the same predecessor gain in probability in equal amounts totalling to the probability of the deleted rule. Let $p_n = (s_n, \chi_n) \in P$ be the production selected for deletion, and $\bar{P}(s) \subset P$ be the set of stochastic productions with $s \in V$ as their production predecessor. Let $Q = \bar{P}(s_n) - \{p_n\}$ be the set of productions whose probabilities need to be changed. Each production $p_k \in Q$ mutates to p^*_k, requiring the associated probability to be changed according to:

$$\pi(p^*_k) = \pi(p_k) + \frac{\pi(p_n)}{|Q|}, \forall p_k : p_k \in Q$$

E.g.: $\left\{ a \overset{0.4}{\rightarrow} ab, a \overset{0.3}{\rightarrow} bb, a \overset{0.3}{\rightarrow} abc \right\} \Rightarrow \left\{ a \overset{0.5}{\rightarrow} bb, a \overset{0.5}{\rightarrow} abc \right\}$,

where $p_n = a \overset{0.4}{\rightarrow} ab$ in this example.

B.4) The probability of a stochastic production may change. This change is selected from the interval $(-\pi(p_n), 1 - \pi(p_n))$, where

$p_n \in P$, is the production selected for this mutation. The addition or difference redistributes probabilities over all the other stochastic productions involving that successor. Assuming the definitions in (B.3) above, and a uniformly distributed random number, σ in the interval $\left(-\pi(p_n), 1-\pi(p_n)\right)$, the new probabilities will be:

$\pi(p_n^*) = \pi(p_n) + \sigma$, and

$\pi(p_k^*) = \pi(p_k) - \dfrac{\sigma}{|Q|}, \forall p_k : p_k \in Q$.

E.g.: $\left\{ a \overset{0.5}{\rightarrow} ab, a \overset{0.5}{\rightarrow} bb \right\} \Rightarrow \left\{ a \overset{0.2}{\rightarrow} ab, a \overset{0.8}{\rightarrow} bb \right\}$, here $p_n = a \overset{0.5}{\rightarrow} ab$ and $\sigma = -0.3$

10.3.2 Parametric Mutation

We now assume parametric 0L-systems, $G = \langle V, \Sigma, P, \omega \rangle$ as defined in Section 5.3.5.1. For the sake of efficiency and ease of implementation, symbols may not gain or loose parameters during mutation. However, new modules[72] may be created with the default number of parameters, or if no default exists, a random number of parameters up to a fixed limit. Productions involving predecessor parametric module may split as follows:

C.1) Productions involving modules with no conditions may split, and gain conditions. Some examples:

$$a(l) \rightarrow a(2l)\, b(l^2) \Rightarrow \begin{array}{l} a(l) : l \le 10 \rightarrow a(2l)\, b(l^2), \\ a(l) : l > 10 \rightarrow a(2l)\, c(l/2) \end{array}$$

$$\begin{array}{l} d(x_1, x_2) \rightarrow d(x_1 + x_2, x_2 + 1) \Rightarrow \\ \qquad d(x_1, x_2) : x_2 > 8 \rightarrow d(x_1 + x_2, x_2 + 1), \\ d(x_1, x_2) : x_1 \le x_2 \mid x_2 \ge 8 \rightarrow d(x_1 - x_2, x_1 + 1) \end{array}$$

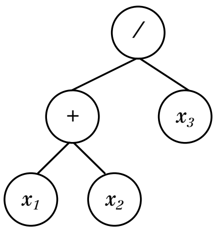
Conditions are created subject to a specialised subset of the parametric mutations D, specified in the next section. This subset consists of a set of binary relational operators $\{\le, \ge, <, >, =, \&, |\}$.

C.2) For productions involving modules with conditions, the conditions themselves may mutate according to the rules specified in the next section.

[72] Recall from section 5.3.5 that a *module* is a *symbol* and its associated *parameters*.

10.3.2.1 *Mutation of Expressions*

Parameters of modules on the successor side of productions are *expressions*. They are parsed into a tree structure and executed during the application of productions. For example, the expression $(x_1 + x_2)/x_3$ can be represented:



The mutation operations outlined here equate to manipulation of such tree structures. Let us assume a predecessor module α from a given production $p_n \in P$, is undergoing parametric mutation. We also assume α has a series of associated parameters $x_1, x_2, \cdots x_k$. We assume the set of binary operators $\{+,-,/,\times,\wedge\}$ and the unary operator {–}. Each node on the expression tree can be recursively subject to mutation by the following rules.

D.1) If the node is a variable it can mutate to another variable. $x_i \Rightarrow x_j$, where $1 \le j \le k$ and $i \ne j$.

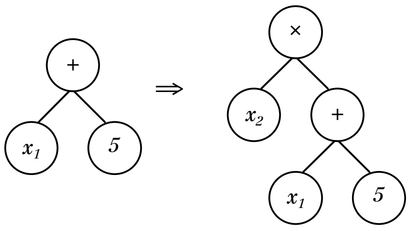
The set of possible variables x_i and x_j are drawn from the set of parameters associated with the predecessor symbol under consideration ($i, j \to [1, n], i \ne j$).

D.2) If the node is a constant, it is adjusted by the addition of some random amount.

D.3) If the node is an operator it can mutate to another operator. Operators must be of the same "arity" (i.e. unary or binary). E.g.: $x_1 + 5 \Rightarrow x_1 \times 5$

A node may mutate to a new expression:

D.4) Nodes may become the arguments to a new expression: e.g.: $x_1 + 5 \Rightarrow x_2 \times (x_1 + 5)$. This example is illustrated graphically below:



D.5) An expression may reduce to one of its operands: e.g.: $x_1 + 5 \Rightarrow x_1$

In the current implementation, only simple arithmetic operators are supported (addition, subtraction, division, multiplication, negation and power). Other functions (such as trigonometric functions) could be added if required. Formal parameters on the left side of productions do not mutate, as this serves no useful purpose.

10.3.3 Development Function and Module Age Mutation

If a module is timed it must have a birth age (α) and terminal age (β). Both these values must be constants.[73] Both constants can be mutated by the addition of a random value. The range of the random value is usually proportional to the size of the constant.

The *development function* (defined in Section 8.2.1) for a timed module controls the behaviour of that module over its growth period. It is expressed in the form $g_s(\lambda_s, \tau_s, \beta_s)$, where *s* is the symbol associated with the function, λ_s the parameter vector for *s*, τ_s is the current age and β_s is the terminal age. A development function could be: $g_s = \tau/\beta$, which is a simple linear function increasing from α/β to 1 from birth to terminal age. Since growth functions are expressions, their internal and external construction is the same as for a module's parameter expressions. Thus the mutations are identical to those described for expressions in the previous section.

10.3.4 Mutation Probabilities

Different sorts of mutations occur with different probabilities. In this case, the term *mutation probability* means the chances at a given time that a particular mutation will occur. Mutation probabilities are expressed as floating-point numbers with the interval [0,1]. A probability of 0 for a particular mutation means that it will never occur; 1 means it will always occur; any value in between sets the overall frequency of occurrence (e.g. 0.1 means, on average, 1 mutation every 10 times the mutation is considered). Some mutation probabilities are dependent on the size of the genotype being mutated — with fixed

[73] In non-evolutionary versions of the system, birth ages can be expressions. See Section 8.2.

mutation probabilities, more mutations will occur on larger genotypes than on shorter ones.

An important consideration for successfully evolving structures is correctly setting mutation probabilities as the evolution progresses. For example, it is better to set production mutation probabilities to maintain or slightly shrink current production size. If rule mutation is biased towards adding productions and/or modules then genotypes tend to become larger without necessarily producing phenotypes of greater fitness. Large sets of productions take longer to parse and in general, take longer to generate.[74] This does not stop rules *evolving* complexity by selection. Rules that take more than a preset amount of time to complete generation are automatically stopped by the system and removed.

The user can change mutation probabilities interactively during the evolutionary process. A wide variety of mutation probability controls are provided in the system developed (see Figure 10-3). This affords a useful aid when one appears to be approaching the desired result and wishes to limit mutations to specific areas.

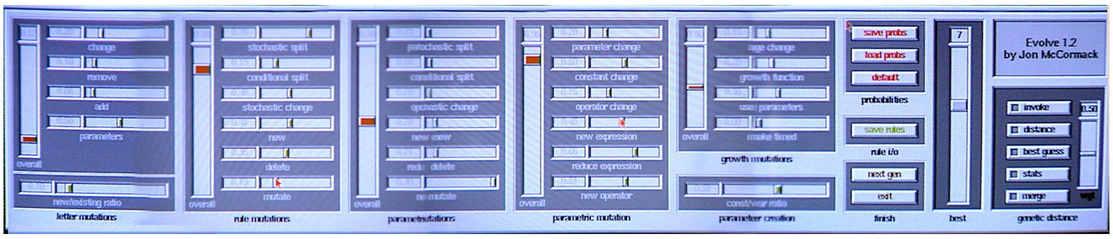


Figure 10-3: Detail of the main mutation parameter controls of the system.

Hierarchical controls also permit changing groups of associated parameters as a whole, while maintaining individual ratios between the associated parts.

10.4 The Interactive Process

To evolve forms interactively we begin with a germinal genotype as described in Section 10.1.4. This genotype may be drawn from an existing library of L-systems, defined by the user, or an "empty" L-system consisting only of identity productions can be used. The subset of the alphabet of the current L-

[74] Although a production as simple as $a \to aa$ doubles the size of the produced string at each derivation (assuming an axiom of a).

system suitable for mutation, ψ, (Section 10.3.1), can be specified at this time. A list of external surfaces to be used is also supplied. The parent production set is then mutated according to probabilities specified. The axiom is mutated in a manner similar to that of productions. After mutation, the L-system is parsed and derived to a specified level, or time-period in the case of timed models. The user may interrupt this process at any time. The software will automatically interrupt the process if the computation time exceeds a specified limit. In traditional uses of the genetic algorithm, population sizes can be large as the selection process is automatic, however in the case of this type of system, selection is based on the subjective notion of aesthetics. This is one reason why the population size in this case is limited. A much more overpowering reason is the limitation of space on the screen to display phenotypes and the computation time involved in generating large populations.

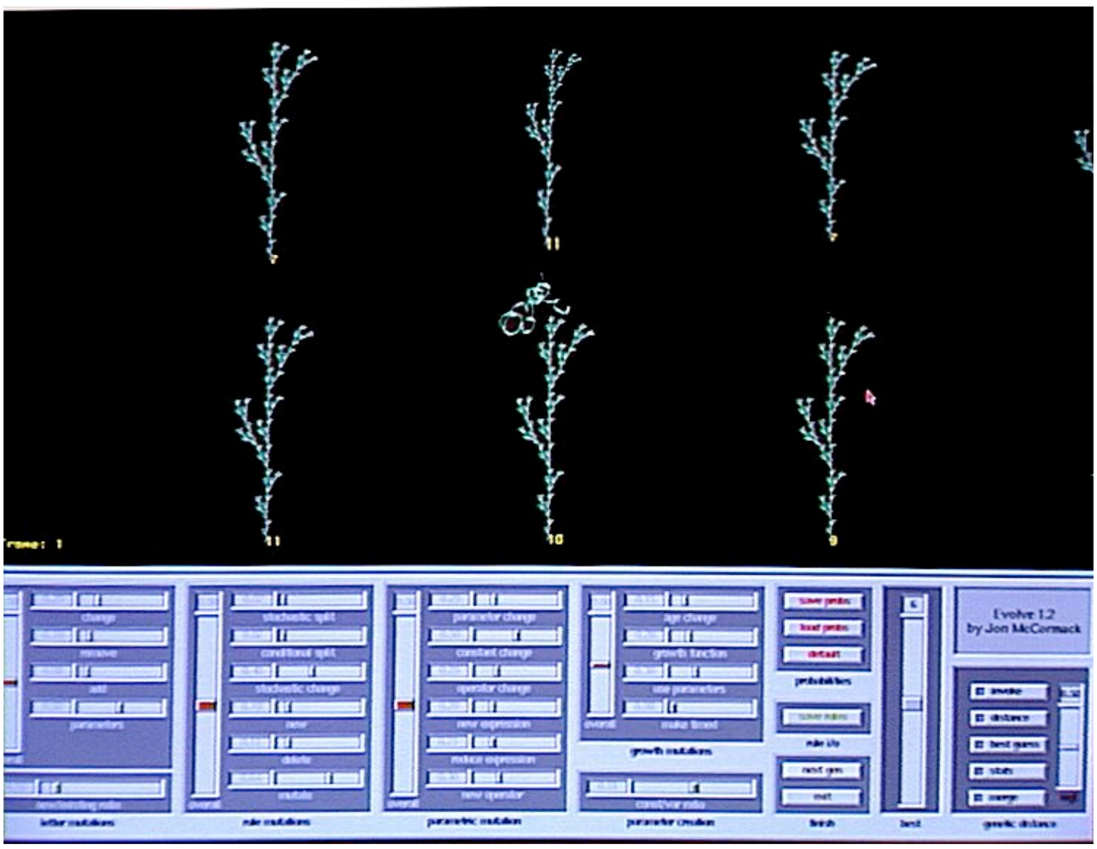


Figure 10-4: Screen shot showing a selection of mutated phenotypes (main area, top) and a selection of the controls for mutation probabilities, fitness selection and genotype file operations.

Usually around 16 mutations per generation are performed. This provides a trade off between generation time and variety. With increased computational power (or more patience) larger populations can be created. The screen size also limits the number of phenotypes that can be displayed and manipulated simultaneously. The parent phenotype is displayed in the upper left-hand corner of the screen followed by its mutated children (Figure 10-6).

At the conclusion of the mutation and generation process, the user may interactively manipulate and examine the phenotypes in space and over time (playback forward, backward, and loop). At some stage, the user decides which phenotype is the most suitable and this becomes the new parent. Selecting the existing parent provides more mutations if none of the current generation are deemed suitable. At any time, the genotype (L-system) for a particular phenotype may be saved to disk as a text file. This file can later be used as a parent for further mutations or generated with a higher degree of accuracy for final output. Sample output in each form is show in Figure 10-5 below.

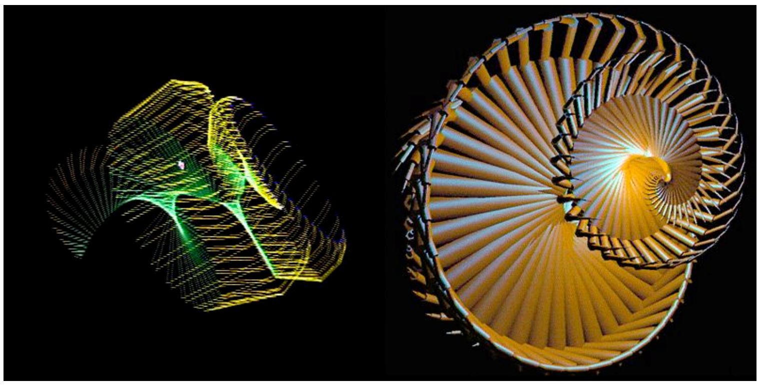


Figure 10-5: A phenotype evolved using the software system described in this chapter, shown in vector format (left) and software rendered with extra surface detail (right).

The mutation/generation/selection process is repeated until a satisfactory form is achieved, or the user runs out of time or patience. A parent phenotype (top left) and 14 child mutations are shown in Figure 10-6. Note the structural variety produced from relatively high mutation rates. Models that incorporate dynamic components (timed L-systems) are played back as animated sequences.

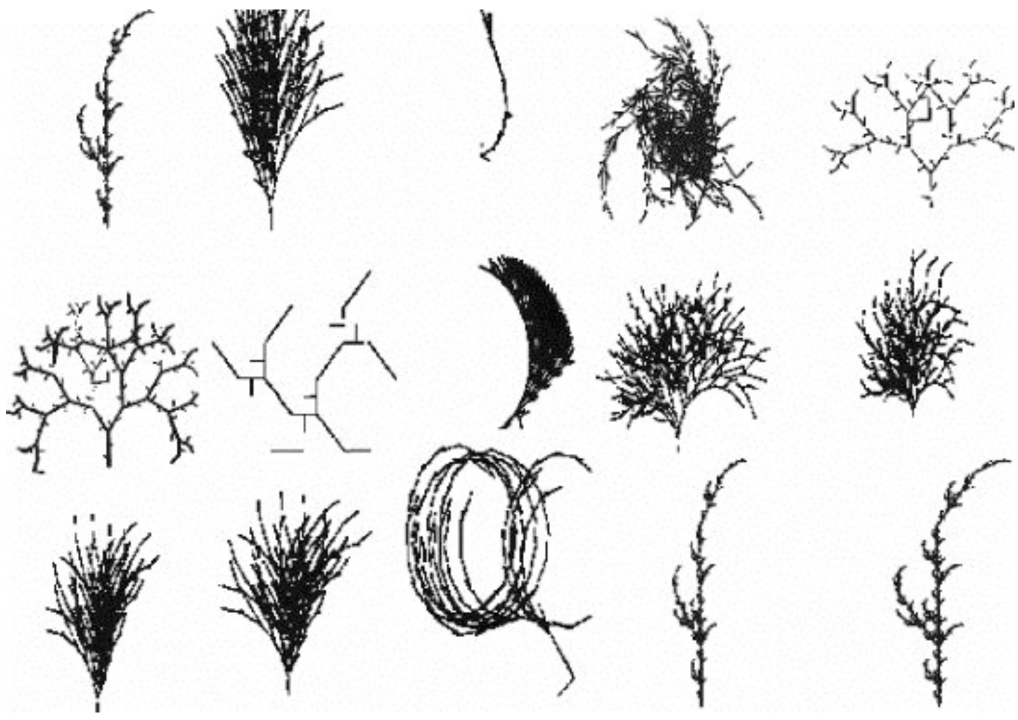


Figure 10-6: Parent (top left) and 14 mutations.

10.4.1 Implementation

The system has been implemented in the C programming language, on Silicon Graphics workstations under the IRIX operating system. The workstation's high-speed graphics performance permits real time previewing of interpreted strings. Since models evolve over time, animation may be previewed as well. A display sub-system accepts graphics primitives on a per-object/time basis, that is, a new display list is created for each frame in a sequence. While the models are defined continuously over time, they are usually sampled at regular discrete intervals for playback. For the case of video animation this is either 25 times per second for frame rendered animation or 50 times per second for field rendered animations.

The generation of several seconds of animation for 16 or more phenotypes can be quite time consuming and thus for preview purposes the samples are taken at wider intervals. For complex geometries, simplified representations of complex surfaces can be substituted to increase display speed.

Once generated, the models are stored in the workstation's graphics memory as display lists. This enables to user to examine the generated models from any position or angle, play the development either forward or backward. Once the most suitable phenotype is picked, the display lists are cleared and a new set of phenotypes are created. At any time, the user can save the L-system rules for a particular resultant phenotype to a human readable file. This can be used to regenerate the model at a later time or as a genotype for further mutation and evolution work.

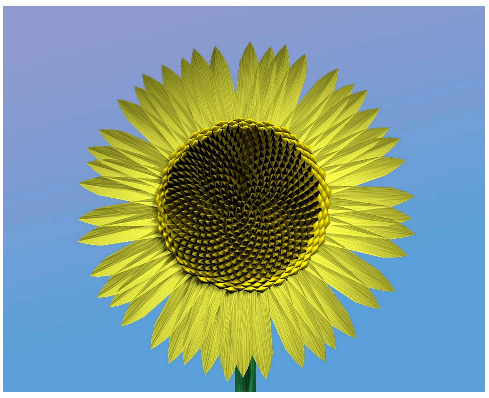
Even simple rules can produce highly complex models that cannot be displayed in real time by the workstation graphics hardware. An option exists to save geometry to disk for rendering by a software renderer. Integrating with the animation system was a key goal in development and the system takes advantage of the features of a powerful existing animation and rendering system.

10.5 Results

To date the system has been used to produce a variety of successful works, both still and animated (McCormack 1992b, 1992a, 1994a). Figure 10-7 shows an example of the results achieved with the system. Starting with the figure shown on the left (**A**, a common sunflower head), after more than 50 generations of aesthetic selection the resultant form is shown on the right (**B**). The key productions that generate each form are shown underneath the corresponding image. In the interests of clarity, some control and material symbols have been removed from the productions shown.

The model also includes morphogenic growth, and the resulting animation is smooth and continuous from birth to the fully developed model (Figure 10-8). The emphasis on the system has been one of a sculptural tool for modelling and evolving growth, form and behaviour.

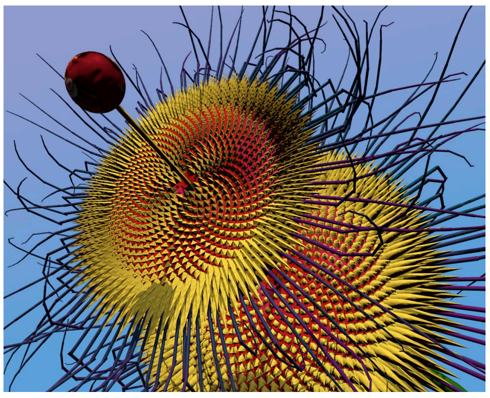
Further images generated by evolved L-systems using the techniques described here can be found in Figure 10-9 at the end of this chapter.



A

```
surface stamen;
surface floret;
surface floret2;
surface leaf2;

productions:
A(n) -> +(137.5)[f(n^0.5) C(n)]
        A(n+1);
C(n) : n <= 440 -> floret
     : n > 440 & n <= 565 ->
                   floret2
     : n > 565 & n <= 610 ->
              ^(90) S(0.3) leaf2;
axiom:
        stamen A(0);
```



B

```
surface stamen;
surface floret;
surface floret2;
surface ball;

productions:
M1(l,s) : l < 2 ->
  [S(s)[f(-1.0*s)stamen]
  F(1.60*s)&(90)A(0)]!(s*0.07)
  F(4.14*s)M1(l+1,s * 0.67);
        : l >= 2 -> [ballh];
A(n)  : n < 1000 -> +(137.5)
  [f(1.10*n^0.5)^(90-(n/1000*90))
  C(n)]A(n+1)
      : n > 1000 -> ;
C(n)  : n >= 10 & n < 600 : ->
       floret2
      : n >= 600 & n < 900 : ->
       floret
      : n >= 900 : -> [
      M2(n,0,1.76,90.40,0.05,3)];
M2(p0,p1,p2,p3,p4,p5): p0<8 : ->
/(p5,p0)^(p5,p0)!(p4)
f(-p2/9)F(p2,p0 + 0.07)^(p3,p0)
M2(p0,p1+1,p2/1.4,p3/2.0,p4*0.8*
   p5,p5*1.8);
axiom:
      M1(0,2.11);
```

Figure 10-7: (**A**) Original form (a Sunflower) and (**B**) the form after many generations of aesthetic evolution using the system described in this chapter. The L-systems generating each model are shown below the image.

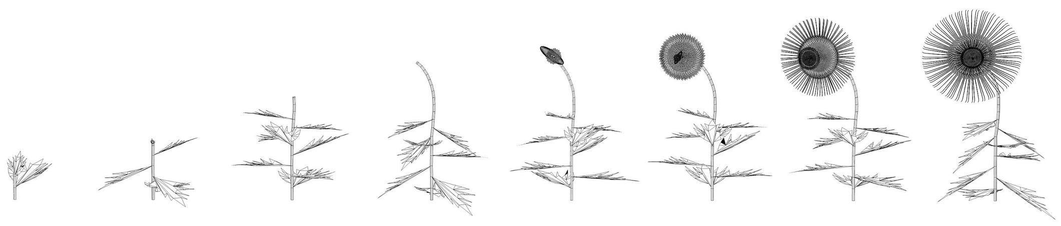


Figure 10-8: Sequence showing the temporal development (left to right) of the evolved model shown in Figure 10-7B. Each element in the sequence is approximately 1 second apart.

10.6 Conclusions and Further Work

The interactive evolutionary technique provides advances in two areas: firstly, it enables a synergy between human and machine. Many of the models and results created by this technique would be extremely difficult, if not impossible, to have been created by explicit writing of rules.

Secondly, the technique allows novice users to create highly sophisticated models with little or no knowledge of the underlying processes involved. Users do not need to learn or understand how L-systems work or write productions, a simple aesthetic selection is the all that is required.

One current limitation of the technique is the speed of generation of phenotypes. Sixteen or more animated models must be generated in a relatively short space of time in order to genuinely call the system "interactive". Simplified surfaces and wire-frame representation help to minimise display time. One would expect as workstation hardware performance increases larger and more complex populations could be generated within realistic timeframes.

As a modelling language, L-systems have large scope in the type of models they can represent. However, a fundamental limitation of the current system is that the representation via L-system symbols is highly topological. Several extensions to allow greater control over surfaces, geometry and texture are currently being tested. Greater control over constraints and physical effects such as light and weather could also improve the system, as would the incorporation of context sensitivity into the set of possible mutations.

10.7 Epilogue

Most of the material in this chapter was developed in 1991 and subsequently published in (McCormack 1993). At that time, the primary work in interactive evolution had been carried out by Dawkins, who, with his *Blind Watchmaker* software evolved two-dimensional, insect-like shapes composed of lines with a designated bilateral symmetry (Dawkins 1986). Following Dawkins publication, Todd and Latham interactively evolved CSG-based forms with their *Mutator* software (Todd, S. & Latham 1991).

Sims describes the use of interactive evolution for computer graphics purposes, using it to evolve morphogenic plant-like structures from parameter sets; images and solid textures generated from mathematical functions and expressions (Sims 1991b); dynamic systems of cellular automata (Sims 1991a); and procedural surfaces (Sims 1993). In the case of plant-like structures, a series of parameters describing “fractal limits, branching factors, scaling, stochastic contributions, etc.” was used to describe the model, in a similar manner to algorithms described by de Reffye (de Reffye et al. 1988). These parameters were used to generate three-dimensional tree structures consisting of connected line segments. More advanced geometry was constructed from the segmented models (using cylinders for example) as a post-processing operation. Growth parameters were also incorporated into the model, allowing animations to be produced (Sims 1990a).

Chen and Lienhardt describe a system to evolve surface forms based on combinatorial maps and deformations (Chen & Lienhardt 1992). This work began with the goal of modelling plant leaves (Lienhardt 1988) and general developmental models for computer graphics. The work is significant as it allowed the development of structures with varying topology over time (i.e. temporal development was achieved by switching between a sequence of modular maps).

The work described in (McCormack 1993) was (to the best of the author’s knowledge) the first published example where interactive evolution had been applied to L-system grammars. The evolution of expression trees is similar to the *genetic programming* techniques of Koza, who evolved LISP s-expressions, to solve a variety of programming problems (Koza 1990, 1992).

10.7.1 More Recent Work in Evolving L-systems

Jacob presents an evolutionary technique for parameter-less, bracketed D0L-systems using a hierarchical, typed, expression system, similar to Koza’s genetic programming method (Jacob 1994, 1995, 1996). An initial population is created from a pre-defined pool of structures, as opposed to a collection of terminal nodes and operators. His system does not use aesthetic selection, rather defines explicit *fitness functions* to rank phenotypes at each generation. A fitness function described seeks to maximise the tree end-points to lie outside a cube of given dimensions.

Ochoa describes a similar technique to evolve parameter-less, bracketed D0L-systems, using genetic programming techniques (Ochoa 1998). She defines mutation and crossover operations achieved by string manipulation of rules. The genetic algorithm used a *steady-state selection* (Mitchell 1996), with only 1/5 of the population replaced at each generation. The fitness function in this case was based on previous studies by Niklas, oriented to the synthesis of realistic tree topologies (Niklas 1982, 1986, 1997). Fitness criteria included positive phototropism, a balanced bilateral symmetry, light-gathering ability, structural stability and proportion of branching points. The results were limited to simple two-dimensional structures composed of lines.

Mock also describes a system to evolve parameter-less D0L-systems using similar techniques to Ochoa (Mock 1998). He uses interactive evolution techniques to select phenotypes for mutation and crossover operations, with special criteria to avoid generating strings with unbalanced bracketing. This is achieved by selecting groups of strings for crossover within sets of bracket pairs ("[" and "]") or without any bracket symbols at all. He also describes experiments with simple automated fitness criteria to evolve simple plant-like structures to volumetric or proportional criteria.

Extending the techniques of Ochoa and Mock, Kókai, Tóth, and Ványi describe a procedure for evolving parametric D0L-systems to fit an existing morphological description (Kókai, Tóth & Ványi 1999). They began with a tree description and were able to evolve a parametric D0L-system to describe it. In the case of this method, the object of maximum fitness must exist, although some attempt can be made to evolve towards a minimal description (i.e. remove redundancy) using D0L-systems that still adequately describes the model under consideration.

Traxler and Gervautz also describe a technique to evolve parametric D0L-systems, based on a previously developed method that combines CSG operations with parametric D0L-systems. (Gervautz & Traxler 1994; Traxler & Gervautz 1996). Their technique for evolution of L-systems is based on that of (Sims 1991b) and restricts genetic operators to the numeric parameters of symbols. Hence, topological changes cannot be achieved by evolution.

Curry describes a system to evolve parametric L-systems using aesthetic selection (Curry 1999). The system described uses seven floating-point numbers in the genotype, representing parameters to a fixed L-system. These parameters control qualities such as branch trajectories and child branch lengths. Mutation and crossover operations are provided, but since the L-system itself is not evolved (only the parameters to specific symbols), the range of structures possible is limited to specific tree-like models.

Hornby and Pollack (Hornby & Pollack 2001a) also evolved parametric D0L-systems to generate "virtual creatures", similar to those of Sims (Sims 1994a, 1994b). Their models were composed of segments connected together via fixed and articulated joints. The strings generated by the L-system were interpreted by turtle commands representing construction operations. These articulated structures are subject to a "quasi-dynamic simulator" to provide physical constraints and give some realism to the physical simulation. For the mutation genetic operators, they use a similar technique to the one described in this chapter, applying special constraints in string creation and replacement to ensure correct bracket closure. They also define simple crossover operations for symbols with multiple productions. Fitness functions were set to maximise the distance travelled by the creature's centre of mass.

Bian Runqiang and colleagues describe a method for automating the inference of L-systems to particular tree species (Runqiang et al. 2002). To avoid the problems associated with random mutation and specification of individual production successors, they use a "repair mechanism" to correct genotypes before fitness evaluation. The repair mechanism divides symbols into subsets, largely based on their turtle interpretation. Only certain combinations of successor strings from each subset have plausible semantics in the context of tree generation, hence there is a need for some form of constraint in the ordering and number of symbols generated in the successor. Strings are modified by the repair mechanism to correct strings with inappropriate syntax. This could be considered as a wider analysis of the bracketing problem, adding the problem of *redundancy* (i.e. strings such as "[-]" have no effect on the generated phenotype).

Other applications of evolutionary programming and L-systems include architectural structures (Coates, Broughton & Jackson 1999; Jackson 2002), tables (Hornby & Pollack 2001b) and neural networks (Kitano 1990).

10.7.1.1 Observations Regarding Evolution and L-systems

Surveying the algorithms and applications described in the previous section, a number of key points can be made regarding the evolution of L-systems.

Most authors limit themselves to evolving D0L-systems, as these are suited to the most general evolutionary methods. For context sensitive, stochastic and timed L-systems, it is more difficult to devise evolutionary representations suited to genetic operations.

Inevitably, due to the operation of bracketed L-systems, measures must be put in place to ensure mutation or crossover operations maintain the bracket balance. This is usually achieved by *(i)* limiting mutations to exclude changing brackets; *(ii)* selecting crossover points and sub-string groups of successor symbols to lie within bracketed pairs; *(iii)* modifying the string to eliminate mismatches and redundancy.

Explicit fitness functions seem difficult to define with any degree of generality. In most cases, the criteria for fitness are simplistic or limited to highly specific domains (such as trees with certain volumetric or topological properties). Not surprisingly, most of the evolutionary applications of L-systems are confined to the application to which L-systems are most commonly deployed — the description of branching tree structures and herbaceous plants. The work presented here remains one of the few applications of L-system evolution to the more general modelling of animated form. Aesthetic selection remains the most general form of fitness criteria, but this generality comes at a cost. This issue is examined more closely in Section 10.7.4.

10.7.2 Other Work in Aesthetic Evolution

Aesthetic evolution (also known as *interactive evolution, artificial evolution,* or *aesthetic selection*) has now become a popular technique that replaces a machine evaluated fitness function with the subjective criteria of the human operator. Since the earlier publications (on which this chapter is based), the use of aesthetic selection has been adopted for a variety of purposes by various authors. These include Rooke, who evolved images generated from mathematical expressions, based on the work of Karl Sims (Rooke 2002). Rooke used a larger set of functions and included a number of "fractal" algorithms in his set of base functions. Graff and Banzhaf used similar techniques to also evolve images, extending the domain to include three-dimensional, voxel images as well (Graf & Banzhaf 1995).

Ventrella generated animated figures by aesthetic evolution using a predetermined topology (Ventrella 1995b, 1995a). He used a "qualitative" physics model as a constraint for the system. Evolved forms had to fit within the constraints of this ad-hoc physics system and the topology permitted by the software design.

Bullhak evolved musical patterns and structures (Bulhak 1999), presenting the user with a small selection of beat-based music clips generated using an evolvable finite state automata (FSA) based system.

Other applications of aesthetic selection include line drawings (Baker & Seltzer 1994), sculptural "art" formed by distorted surfaces of revolution (Tabuada et al. 1998), and architectural applications (Rosenman 1997; Soddu 1998).

10.7.3 The Use of Crossover Operators

One popular component of genetic algorithms is in the use of *crossover* operator[75] to combine sections of the genotypes of two or more parents (Holland 1992, page 97). Dawkins' *Blind Watchmaker* software did not include a crossover operation, but many systems since that time have. The system described here did not use crossover, although some attempts at incorporating it in subsequent versions of the system were abandoned due to the difficulty of controlling how crossover affects the resultant phenotypes. The system has many controls to adjust mutation probabilities for each mutable component of the system (see Figure 10-3). Deft use of these controls allows the operator to control which components of the phenotype are mutated. This permits a fine degree of control during latter stages of the evolution, where the form is "almost right" and the operator wishes to avoid drastic changes that may be

[75] Holland and some other authors refer to this as the "crossing-over operator", but in much of the modern literature, both computing and biological, it is simply referred to as "crossover".

brought about by, for example, structural changes to interdependent rules (see Section 10.7.4.2 also).

10.7.4 Aesthetic Evolution and Subjectivity

Typically, genetic algorithms evolve towards finding maxima in *fitness*, where fitness is some criteria that can be evaluated for each phenotype of the population. Many systems define an explicit *fitness function* that can be machine evaluated for every phenotype at each generation (Mitchell 1996). In essence, genetic algorithms are a search technique, seeking to find maxima in the *fitness landscape*.

Regardless of the system or form being evolved, aesthetic selection relies on the user to explicitly select phenotypes at each generation. Users typically evolve to some subjective criteria — often described as "beautiful", "strange" or "interesting" — criteria that prove difficult to quantify or express in a machine representable form (hence the use of the technique in the first place).

Whitelaw looks at modern-day successors to Latham and Sims who use artificial evolution techniques, principally for creative purposes (Whitelaw 2002). He contrasts the work of Steven Rooke with that of Dutch artists Driessens and Verstappen. He sees the work of Rooke following a direct lineage from the methods pioneered by Dawkins, Latham and Todd and Sims. Driessens and Verstappen however, *subvert* the evolutionary process (Section 4.6.1). Typically, genetic algorithms search vast spaces, too large or difficult to be searched by other techniques. In terms of impetus, the artist must first construct the potential for such a space to be vast by adding complexity, usually in the form of a variety of operators and functions (in the case of Rooke's images for example).

Rather than looking for this expansive landscape in their art, Driessens and Verstappen define deliberately simplistic representations and automate the fitness criteria offering many variations of basic thematic structures, such as recursive cuboids (Driessens & Verstappen 2001b) or visually organic tuboid structures, reminiscent of some of Haeckel's drawings (Haeckel 1998). Here, the aesthetic selection process becomes one of parody, whereby the machine-defined fitness function produces phenotypes of endless variety, but aesthetic banality. The process of evolution itself becomes the primary focus of artistic exploration, rather than the results that the process produces.

Dorin criticises the concept of aesthetic evolution from a creative perspective. He argues that the process of selection is limiting, likening it to "pigeon breeding" and that the real artistic merits lie in the development of the model that is being mutated, rather than the aesthetic selection of phenotypes generated by that model (Dorin 2001a).

10.7.4.1 Problems with Aesthetic Evolution

In terms of being an efficient search technique, aesthetic evolution has two significant problems:

- The number of phenotypes that can be evaluated at each generation is limited by both screen area (in the case of visual representation) and the abilities of people to perform subjective comparisons on large numbers of objects (simultaneously comparing 16 different phenotypes is relatively easy, comparing 10,000 would be significantly more difficult, with $O(n^2)$ comparisons to consider).
- The subjective comparison process, even for a small number of phenotypes, is slow and forms a bottleneck in the evolutionary process. Human users may take hours to evaluate many successive generations that in an automated system could be performed in a matter of seconds.

What we would like is a system that combines the ability to subjectively evolve towards phenotypes that people find "interesting" without the bottleneck and selection problems inherent in aesthetic evolution. This problem is addressed (in a certain context) in (McCormack 2002) where the actions of the audience of the artwork are used to control the evolutionary process implicitly based on the interest they shown in the artwork.

10.7.4.2 Dependency Relationships

One particular difficulty[76] of the system described here is the way sets of rules may become "hidden" for many generations only to resurface some time later. This is due to dependency relationships and the dynamic nature of the gram-

[76] Or maybe benefit, depending on the results required.

mar. For example, consider the following set of productions in a D0L-system and the first four derivations:

L-system		**Derivation**
$\omega:$	a	a
$p_1:$	$a \rightarrow bb$	bb
$p_2:$	$b \rightarrow c$	cc
$p_3:$	$c \rightarrow ab$	$abab$
		$bbcbbc$

Now consider the following mutation:

$$p_2 \Rightarrow p_2^* = \{b \rightarrow b\}$$

The derivation is now:

a
bb
bb
bb
bb

If the current string under derivation, μ, does not contain the symbol c, p_3 will not be called upon. Yet the production continues to exist as part of the system. Some time later, a successor may again mutate to include c, which means that the particular production (p_3 in this case — or a mutated variation) will again come into effect. More complex dependencies (such as a single production that "bridges" two groups of rules) means that a single mutation of the successor in a production can cause major changes in the resultant phenotype.

10.7.5 The Inference Problem

As was observed by Prusinkiewicz and Lindenmayer et. al.:

> Random modification of productions gives little insight into the relationship between L-systems and the figures they generate. (Prusinkiewicz & Lindenmayer 1990, page 11).

The *inference problem* — inferring an L-system from the observation of an existing developmental process — remains difficult. Recent attempts give some insight into how the problem may be automated by genetic algorithms (Runqiang et al. 2002), in the application of tree generation. However, as the system described in this chapter indicates, under limited circumstances the synthesis of models that could *never* be observed (or perhaps even explicitly designed) is possible using the aesthetic evolution techniques described here. Aesthetic selection is a solution to the problem of creative exploration. Certain inferences may be implicitly solved by this technique, but this says nothing about the relationship between representation and model explicitly, other than that the model generates the representation. That is a problem, it seems, that needs addressing via other techniques.

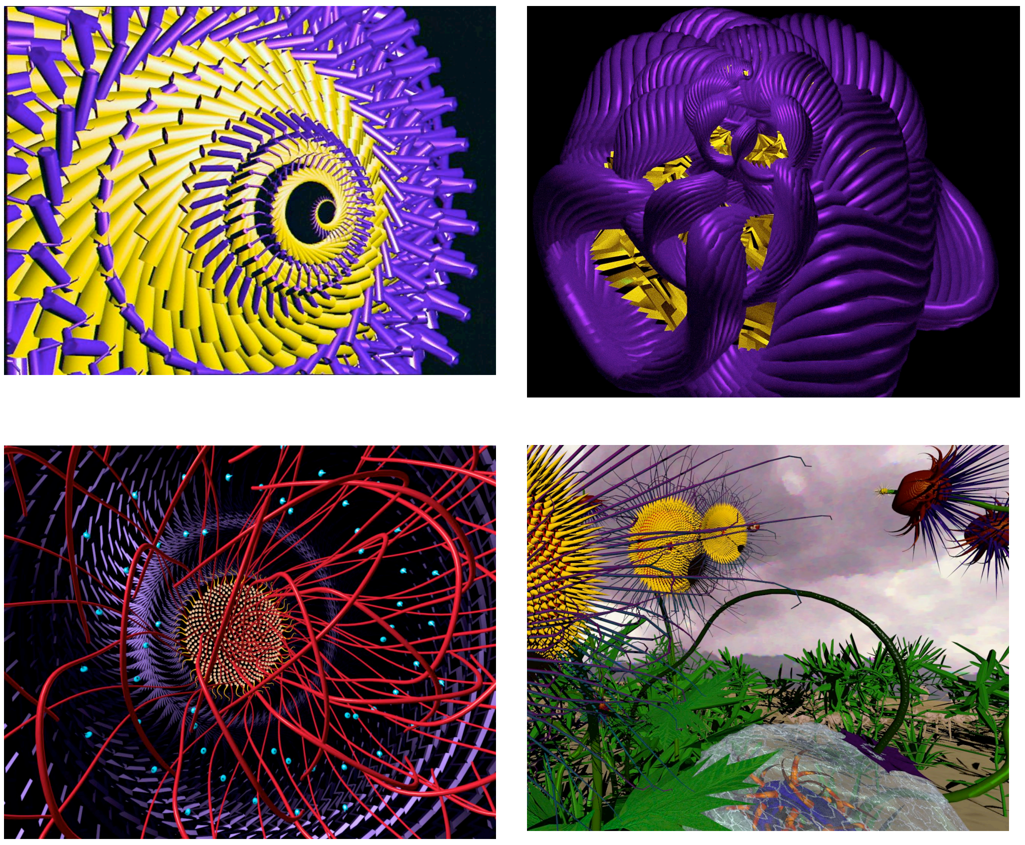


Figure 10-9: Rendered images of models created using the evolutionary system described in this chapter.

11 Music Composition[77]

With the aid of electronic computers, the composer becomes a sort of pilot: pressing buttons, introducing coordinates, and supervising the controls of a cosmic vessel sailing in the space of sound, across sonic constellations and galaxies that could formally be glimpsed only in a distant dream.

—*Iannis Xenakis, 1971*

Thus far, this thesis has primarily focused on the generation of dynamic visual models. Indeed, L-systems are best known as a popular method for the modelling of space filling curves, biological systems, and morphogenesis. In this chapter, the L-systems described in Chapters 5 and 8 and the developmental systems in Chapter 9 are adapted to a system designed for music composition. Representations of pitch and timbre are encoded as grammar symbols, upon which a series of re-writing rules are applied. Parametric extensions allow the specification of continuous data for the purposes of modulation and control. Such continuous data is also under control of the L-system. The developmental model provides timing and synchronisation control in an intuitive way. Using non-deterministic grammars with context sensitivity allows the simulation of Nth-order Markov models with a more economical representation than transition matrices and greater flexibility than previous composition models based on finite state automata or Petri nets. Using symbols in the grammar to represent relationships between notes, (rather than absolute notes) in combination with a hierarchical grammar representation, permits the emergence of complex music compositions from relatively simple grammars.

[77] This chapter is based on material first published in (McCormack 1996).

11.1 Music

Music is the organization of sounds in space and time. It is one of those areas of human activity that, despite its ubiquitous presence in human culture, remains resistant to a detailed analytical understanding. It is an open problem why something so apparently simple as the succession of changing discrete tones has the power to move the listener emotionally. Of all the arts, music is considered "to be open to the purest expression of order and proportion, unencumbered as it is by material media" (Loy 1989). It has been often noted that music bears a close affinity to mathematics, and that notions of mathematical and musical aesthetics may have some similarities (Schillinger 1948). Since the advent of computing, many researchers and musicians have turned to the computer as both a compositional, and synthesis device for musical expression.

This chapter describes a system for *computer music composition*. The majority of this section will focus on the application of computers to the composition of music. This is not to discount the many other uses of computers in music, such as sound synthesis, acoustic and physical modelling, automated notation, score editing and typography. A detailed overview of computer-based applications to music in all these areas can be found in (Roads 1996).

Software for music composition can be broadly categorised into *algorithmic composition* or *computer-aided composition* (Miranda 2001, page 9). With algorithmic composition, the software generates the composition somewhat autonomously; with computer-aided composition, the software serves as a tool to assist the composer develop and organise a composition. Generative compositional systems, hence the work described here, fall into the algorithmic composition category.

A major part of music creation involves the use of certain musical*formalisms* (systematic ordering), in addition to algorithmic and methodic practices. Many of these were well established before the advent of digital computers. In a substantial survey of computer composition, Loy and Abbott suggest that the application of musical formalisms based on a priori theories of composition have been largely developed in the twentieth century, originating with composers such as Schoenberg and Hindemith (Loy & Abbott 1985). Loy and Abbott also note the contribution of music typographers and instructors who have found formal systems attractive to their profession.

Supper divides algorithmic music composition into three distinct categories (Supper 2001):

1. Modelling traditional, non-algorithmic compositional procedures;
2. Modelling new, original compositional procedures, different from those known before;
3. Selecting algorithms from extra-musical disciplines.

According to these classifications, the system described here is from category 3.

11.1.1 Musical Representation Systems

Representations for notation and programming of music (and thus computer composition) can be broadly classified into *symbolic* and *iconic* (Loy 1989). Symbolic representations relate a symbolic and sonic event, but have no direct visual resemblance to the sound. Iconic representations on the other hand, carry some resemblance (usually visual) to the sound the icon represents. Common music notation combines both forms of representation (time signature, clefs, bar lines are symbolic; pitch coding, crescendo, diminuendo are iconic). While this system is not completely adequate, particularly for some contemporary forms of music, it is an extremely versatile and capable notation for a diverse range of musical expression in the western tradition.

Many other representation schemes have been used, related to a particular musicology and musical culture (Copland 1988), or when traditional notation systems have proved inadequate. For example, Wishart devised a number of complex notation schemes to describe and articulate compositional works for the human voice, with an emphasis on the continuous nature of sound over space, pitch and timbre as opposed to the more discrete and “grid-like” thinking behind conventional schemes (Wishart 1996).

Iannis Xenakis was a major figure in formalised systems for computer music composition (Xenakis 1960, 1971, 1992). He developed many representational systems unique to his application of stochastic process in composition. Representation and notation systems particular to generative processes often mirror the formal structure of programming languages. A number of composers have developed generative process notation that revolves around physical actions or instructions that may be open to a wider variety of interpretations than the formal specification of programming languages. Examples of such systems can be found in (Cardew 1972; Nyman 1999).

A work, which typifies the customisation of notation to generative musical process, is that of British composer Cornelius Cardew, which was discussed in Section 3.6.3.

11.1.2 Musical Patterns

It has been observed that almost all forms of music involve repetition (Leach & Fitch 1995), either of individual sequences of notes or at some higher levels of structural grouping. Often these repetitions appear at a number of different levels simultaneously. Some compositions repeat patterns that are slightly changed at each repetition, or revolve around some musical "theme", whereby complex harmonic, timing or key shifts in a basic theme provide a pleasing musical diversity.

It is also commonly known that what allows us to identify an individual piece of music is the *change* in pitch between notes, not the pitch of the notes themselves. We can change the key of a composition (equivalent to multiplying all frequencies by some fixed amount) and still recognise the melody.

The compositional problem has seen a wide variety of approaches. Loy and Abbott's survey paper details many different methods: stochastic and combinatorial models; grammar based; algorithmic; process models; and other models derived from Artificial Intelligence techniques.

The methods used in the system described here are L-system based, developed on the system and formalisms described in Chapters 5, 8 and 9. While grammars have been a popular method for algorithmic composition, in most instances the grammars used are relatively primitive and thus limit the scope of control and diversity of composition that can be generated. The approach used in the system described here, is to take developments in grammars used for modelling of biological morphogenesis and herbaceous plants in computer graphics, and apply them to music composition. In addition to being able to represent a variety of both stochastic and deterministic compositional techniques, the system has compositional properties unique to this approach.

The remainder of this chapter is organised thus: Section 11.2 briefly looks at the problem of computer based musical creativity and explains the rationale behind computer music composition using algorithmic composition methods. Section 11.3 examines algorithmic representations for music composition and introduces the basis of the grammar method. Section 11.4 looks at related work in composition using musical grammars. Section 11.5 details how grammars can be adapted for music composition, with specific emphasis on L–Systems and the developmental extensions and models described in this thesis as the basis for the model. Section 11.6 discusses implementation details. Section 11.7 outlines the use of developmental models for music generation. Finally, Section 11.7.2.1 details possible extensions and further work.

11.2 Musical Creativity

Any composition system must work with the human composer. A complete theory of general creativity, or even musical creativity, remains elusive. While many attempts have been made to study and document the creative process: (Barron 1969; Boden 1994; Crutchfield, R. S. 1973; Dartnall 2002; Schillinger 1948) for example, any generality seems difficult to uncover. In many cases, people do not know how or why they make creative decisions, and much of the creative process is difficult to repeat in controlled experiments.

The question of whether creativity is computable is an issue of even greater controversy. Many researchers from Poincaré (Poincaré 1923) to Penrose (Penrose 1989) have argued against a computable model of creativity because, simply put, the underlying mental processes are not Turing computable. While creativity is clearly an ability of the human mind (and body), to a large extent practical creativity is deeply related to an individual's life experience and of "being in the world" (Dreyfus 1991). A life experience includes relationships to the environment, interaction with both living and non-living things, and social and cultural constructions. We all know that these things have a major effect on a person's mental states. Much creativity also depends on serendipitous and chance events in the external world, both conscious and unconscious. The explicit use of random events in composition is a practice many centuries old. It can be found in the works of many artists – from Mozart and Hayden, to Burroughs and Pollack. Bense's theory of "generative aesthetics" used randomness to replace what in art was described as "intuitive" (Reichardt 1971). Creativity is not only influenced by external events, it may also be stimulated by artificially induced internal effects — some artists claim their best creative work is done under the influence of a large variety of chemical substances.

Even if some day it *may* be possible to build a computer with processing capabilities similar to that of the human brain as some have proposed (Moravac 1988), computing a simulated life experience would appear to be impossible (Oreskes, Shrader-Frechette & Belitz 1994). It is feasible that a robot that has life-experience in the real world may have the potential to exhibit creative behaviour, but again if life-experience is bound to physicality and matter (as opposed to mechanisms and processes) any creativity exhibited by such a robot may not be recognised as human-like creativity, or even recognised as creativity at all.[78]

11.2.1 Semantics and Meaning

Johnson–Laird, has argued that music is an excellent testing ground for theories of creativity, because, as opposed to other Artificial Intelligence domains such as natural language generation, music avoids the problem of semantics (Johnson-Laird 1993). That is, a musical expression cannot be judged to be true or false. While this statement may be literally correct, many musicians would argue that as part of the compositional process, musical expressions *do* contain semantics (Roads 1985). For example, some composers associate emotive, structural or linguistic semantics with individual themes or passages in a composition. Carl Orff described musical composition as "the raw expression of human energies and states of being" (Orff 1967). The musical development of a composition is driven by the associated development of emotions, structure or language (Mandler 1975). In this sense, such compositions do have meaning and there is a semantic association between musical expressions. Such expressions have ordering and denote an emotive symbolic system. According to Langer, music, as a symbolic system of emotional arche-

[78] An alternative view of connectionism and embodiment can be found in (Globus 1995).

types, conveys a “morphology of feeling,” akin to algebraic notation conveying a mathematical expression (Langer 1948).

Some composers (Brian Eno, for example) visualise abstract or real environments as mental images, and the visual and emotive feeling of these environments directly influences the composition. Composition is rarely a one-way process from mind to finished composition either. It is an *iterative feedback* process — hearing the music causes change in the composition. Composers are rarely interested in one aspect of the composition in isolation, such as pitch, timing or timbre; rather these things in total — the “surface” of the music — greatly influence the emotional feeling and thus carry the composition.

In summary, while music may appear to be an easier domain to test theories of creativity, it is more difficult to expect a computer program to exhibit a genuine musical creativity, one approaching that of human composers. A more achievable approach, one adopted by the system described in this chapter, is to use the machine as a synergetic partner to the human composer. The human and computer work in tandem through an interactive feedback process, the computer presenting new musical possibilities by synthesising complexity and variation, the human composer directing the overall creative “quality” of the composition. This human-machine feedback process, known as *computational synergetics* has been used to significantly enhance both scientific (Zabusky 1984) and artistic (McCormack 1994c) creativity. The computer, used in heuristic mode can lead the human collaborator to new discoveries that may have been difficult or impossible without the collaboration. In this way, the machine acts as a kind of “creative amplifier”, enhancing the creative potential of the composer.

11.3 Representation for Music Composition

This section looks in more detail at the various representations for music composition. Since the focus of this chapter is on grammar-based techniques, details on various approaches are deferred to Section 11.4.

11.3.1 Procedural Models

A grammar-based approach to composition makes heavy use of *procedural models*.[79] One advantage of procedural methods in general, is that they allow a terse representation of a complex outcome. Likewise, with generative grammars, there is the possibility for the *emergence* of complexity through the repeated production of simple (and possibly deterministic) rules. One has to be careful with such analogies however, for biological morphogenesis relies also on many factors, such as the laws of physics, special properties of carbon atoms, the *self-organisational* properties of many molecular and cellular structures and complex environmental interactions. Certainly, these features are not built into this music compositional model, though it is possible that some of these concepts, in an abstract sense, could be incorporated into the model.

11.3.2 Stochastic Processes

A diverse variety of *stochastic* processes have been applied to music composition (Jones 1981, Chapter 5; Moore 1990). Most involve two stages. First, some existing musical expression or quantity is analysed. Following the analysis, re–synthesis is applied using the selected stochastic technique. Importantly, while we can undertake statistical analysis of many musical patterns, when we attempt to synthesise music based on that analysis, the results rarely seem to have the clarity or intent of human composition. For example, Voss (Voss & Clarke 1975) analysed many different types of music and found that several were statistically similar to *1/f* noise, however when one converts *1/f* noise to musical notes, the results have little in common musically with any of the compositions from which the statistical analysis was derived. Given the discussion in Section 11.1.2 regarding formalisms, it is clear that a more sophisticated model is required.

[79] The term "procedural model" is sometimes associated with music in relation to techniques that address the issue of rule generation from lower-level phenomena. Used in this sense, the procedural model simulates a listening process to both use and build knowledge grammars, which undergo constant transformation (Roads 1985).

11.3.3 Events and Event Spaces

Generally in composition, one primary concern is the notion of temporal *events* and *event spaces*. The general term *event* is used to represent some basic building block of composition, in many instances a note or set of notes, but in other compositional applications, events may refer to sound complexes, individual sound samples or other basic musical elements. An event space is an ordered set of events. The number of events contained in a given event space represents its *order.*

11.3.4 Markov Models

One popular approach to computer composition is the use of *Markov chains*, where the probability of a specific event *i,* is context dependent on the occurrence of a previous event or event space (Jones 1981). Most attempts using this form of modelling first require some existing composition to be analysed. For the purposes of explanation, we will restrict this discussion to consider only the pitch of notes, however there is no reason why the same techniques cannot be applied to other musical qualities such as duration, volume or timbre. We also consider only *discrete–valued* (as opposed to *continuous–valued*) data. Markov processes are suited to the analysis of conventional musical pitches. A Markov process is considered *stationary* if the transition probabilities remain static.

An *N*–order Markov model can be represented using an *N+1* dimensional *transition matrix*. Let us assume the process under consideration uses *n* distinct events. Elements of the matrix contain probabilities for a new event based (column) on a given event (row). For example, Figure 11-1 shows a sequence of note events, and the corresponding transition matrix. This is a *1st–order model,* where the probability of a given event depends only on the event immediately preceding it. Each element in the transition matrix, $P_{i,j}, i = 1,\ldots n; j = 1,\ldots, n$ represents the probability of event *j* occurring given that event *i* has just occurred. $P_{i,j}$ is calculated by summing the occurrences of each note *j* that follows note *i* in the input melody. Counts in each row are normalised so the combined probabilities for each row, *i,* sum to 1, i.e.:

$$\sum_{j=1}^{n} P_{i,j} = 1,\ i = 1,\ldots,n \tag{11.1}$$

Set of input notes:

E D C D E E E D D D E G G E D C D E E E E D D E D C

Transition Probability Matrix:

Next Event

Current Event

	C	D	E	F	G
C		2/3	1/3		
D	3/10	3/10	4/10		
E		5/11	5/11		1/11
F					
G			1/2		1/2

Figure 11-1: A set of input notes (from a popular nursery rhyme), and the corresponding 1st order Markov model, represented as a transition matrix. Each element of the table P_{ij} is the probability of event *j* given the previous event *i* occurred. Empty elements in the matrix represent a probability of 0.

Jones shows how transition tables can be converted to *event–relation* diagrams, which are essentially equivalent to finite-state automata (Jones 1981). Lyon has used Petri Nets to extend the finite-state model generated by stationary *N*–order Markov models for real–time performance (Lyon 1995).

The Markov process for composition has several problems:

- Unless the transition table is filled with randomly generated probabilities, some existing music must be input into the system. The resulting Markov model will only generate music of statistically similar patterns to that of the input. For short sequences, many elements of the transition table will have a 0 probability. To introduce variety, it is possible to set these entries to some small probability, provided equation (11.1) is satisfied.
- For higher order models, transition tables become unmanageably large for the average computer. While many techniques exist for a more com-

pact representation of sparse matrices (which usually result for higher order models), these require extra computational effort that can hinder real-time performance.

- *Ergodic*[80] Markov chains are preferred for composition since their behaviour is more predictable and manageable from a compositional point of view (Grimmet & Stirzaker 1982).
- Most importantly, while such models have been used successfully in composition, they are limited in the variety of music produced by any given transition table. They also provide little support for structure at higher levels (unless multiple layered models are used where each event represents an entire Markov model in itself).

11.4 Related Work in Grammar Based Composition

This section details related work in grammar-based compositional systems in general, and L-system based systems in particular. Related (but conceptually different) systems such as finite state automata are briefly examined.

11.4.1 Related Work in Formal Grammars

It has been noted that many hierarchical music structures are similar to linguistic structures (Miranda 2001, Chapter 1). A number of composers and programmers have applied grammar formalisms to the *synthesis* and *analysis* of syntactic structures in music (Baroni & Callegari 1984; Bell, B. 1992; Bell, B. & Kippen 1992; Burnstein 1976; Cook, N. 1987; Cope 1991; Holtzman 1980, 1981; Jones 1981; Laske 1975, 1979; Lidov & Gabura 1973; Molino 1975; Roads 1978, 1985). Some of these efforts attempt to relate the work of Chomsky in linguistics to music (Lerdhal & Jackendoff 1983; Winograd 1968).

Grammar systems include both *generative* and *transformative* types. Generative types create sentences (compositions) from initial axioms, whereas transformative types translate from one formal language to another. The systems considered here are primarily generative.

[80] An ergodic Markov chain contains exactly one *recurrent* class and 0 or more *transient* classes. See (Miranda 2001, pages 69-70) for details.

Holtzman's Generative Grammar Definition Language (GGDL) compiler (Holtzman 1980, 1981) is based on Chomsky grammars of type 1 to 3 (regular, context-free and context-sensitive) (Chomsky 1957, 1963). Jones discusses context–free *space grammars* that can operate across many dimensions, where each dimension represents a different musical attribute (Jones 1981). An interesting property of space grammars is that an *n*–dimensional grammar can generate an *n*–dimensional geometric shape, which may provide a higher-level iconic representation of the music generated by a particular set of rules.

11.4.1.1 Related Algorithmic Composition Techniques

Much development in music composition and synthesis has been performed in related areas such as finite state machines (automata) — both deterministic and non-deterministic (Chemillier 1992; Pope 1986); chaos, fractal techniques (Leach & Fitch 1995; Voss & Clarke 1975); stochastic methods (Hiller, L. A. & Issacson 1959; Jones 1981; Xenakis 1960); cellular automata (Beyls 1989; Bowcott 1989; Chareyron 1990; McAlpine 2000; McAlpine, Miranda & Hoggar 1999); neural networks (Dolson 1989; Gjerdingen 1989; Scarborough, Miller & Jones 1989; Todd, P. M. & Loy 1991); procedure-oriented and recursive composition. Moore also gives an overview of some of these methods (Moore 1990).

Surveys of grammar-based approaches to algorithmic composition can be found in (Hiller, L. 1981; Roads 1996) and various algorithmic composition methodologies are detailed in (Roads 1996, Chapters 18 and 19) and (Miranda 2001). Chapter one of David Cope's book *Computers and Musical Style* provides a comprehensive background and overview of composition systems based on generative grammars (Cope 1991, Chapter 1).

A number of evolutionary techniques have also been applied to algorithmic composition systems. See, for example (Bentley & Corne 2002; Horner & Goldberg 1991; Miranda 2001, Chapter 6; Todd, P. M. & Werner 1998; Wiggins et al. 1999) for some overviews.

11.4.2 Other Work in L-systems Composition

A number of other authors have developed generative musical systems based on L-systems and related techniques. These are most closely related to the system described here. However, most applications have been kept relatively simplistic, generally using DOL-systems, with a basic interpretation.

Prusinkiewicz and Hanan describe a simple system whereby an L-system is used to generate a two-dimensional space-filling curve, which is then traversed to generate pitch sequences (Prusinkiewicz & Hanan 1989). Horizontal lines are mapped to pitch and the length of the horizontal line maps to duration. This process is illustrated in Figure 11-2. Similar methods were also described in (Prusinkiewicz 1986a).

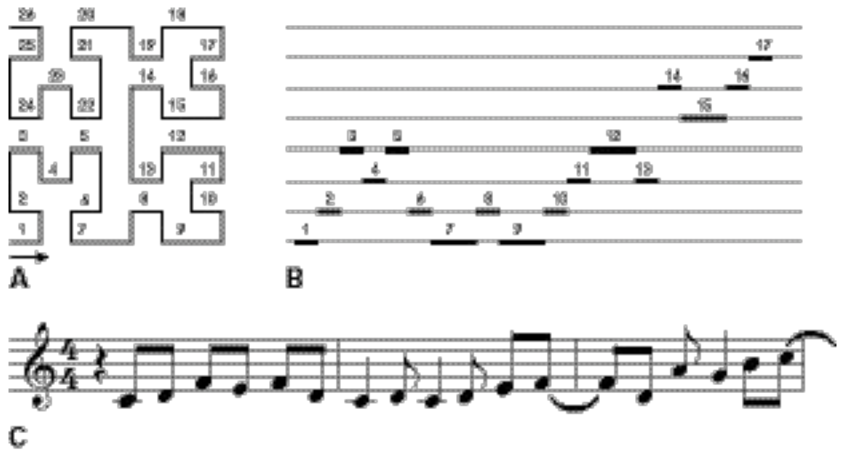


Figure 11-2: Musical interpretation of a Hilbert curve generated by an L-system [from (Prusinkiewicz & Hanan 1989)]. The curve (**A**) is traversed in the order shown starting in the direction of the arrow. The hight of the curve is mapped to pitch, length of horizontal lines to duration. (**B**) shows a "piano roll" representation with pitch in the vertical axis and time on the horizontal. The first three bars of the final score are shown in (**C**).

Kyburz generated music with L-systems by letting symbols in the derivation string map to "musical objects" (small sequences or motifs). The derivation proceeds for a certain number of steps, at which time the resultant string defines the order of musical sequence (Supper 2001).

Based on ideas from (Mason & Saffle 1994) and the previously published work described here (McCormack 1996), Soddell and Soddell used an L-systems description of microbe structures as a basis for musical interpretation (Soddell & Soddell 2000). Their technique is used as ***sonification***[81] providing an auditory distinction between different models. Changes in direction of growth (determined by

[81] Sonification is the sonic equivalent of visualisation, whereby data is mapped to sound in order to analyse and explore that data from an aural perspective.

branching angles) were mapped to changes in pitch, and distances between branches to note duration.

Other examples of L-systems music generation can be found in (Langston 1986, 1990; Nelson 1996; Sharp 1998).

11.4.3 Characterisation of Music Generation with L-systems

Most of the systems described in the literature are relatively simple, relying primarily on D0L-systems with a limited musical interpretation. Few controls are provided to address important musical qualities such as timbre, dynamics, and complex timings. The discrete nature of the underlying model makes it difficult to enhance the compositional flexibility and musical gamut of such systems. Interestingly, too, is that a number of authors start with specific L-systems created for non-musical purposes, such as space-filling curves or biological models, and use these non-musical starting points as the basis for generating music. This may be one reason why Supper classifies L-system music generation as coming from "extra-musical disciplines". However, the use of non-musical material and influences has often found favour from composers.

In the system developed in this chapter, a number of the limitations of previous grammar–based models are addressed. These limitations include: the discrete nature of symbolic grammars; lack of continuous control and modulation of properties; deficiencies in music representation, such as the use of fixed or simplistic timing and duration specification, lack of polyphony, lack of timbre control; and the incorporation of semantics into a grammar-based compositional system. The ways in which these limitations have been addressed with a number of new developments is described in this chapter. A multi-player architecture permits multiple parts and timbres within a single performance (Section 11.6.2). A syntax for polyphonic note specification combined with a novel context interpretation uses past polyphonic and temporal states to determine rule matching criteria (Section 11.6.3). The use of timing symbols, indexed duration tables, and interpretative state modification by produced symbols gives a parametric L-system grammar the ability to modify and control note timing (Section 11.7.1). The parameterisation of timed modules permits continuously variable properties to be incorporated into a composition alongside the discrete event-based nature of "traditional" D0L-system models. The use of a hierarchical model (Section 11.5.5) more easily permits the authoring of complex musical structures, in ways in which music is often thought of (Buxton et al. 1978, 1985).

A criticism of grammar-based representations is in their fundamental separation of syntax and semantics (Roads 1996, page 893). These criticisms relate in particular to using grammar representations for theories of music cognition. There is some doubt that there exists a musical "deep structure" in the sense of Chomskian linguistics, and that syntactic structures have no strict correlation to semantic context (Baroni & Jacoboni 1978). It has been argued that imposing hierarchical structure neglects the multitude of possible ways in which non-hierarchical parsing may be applied (Minsky 1965, 1981; Roads 1985). In Minsky's view, the grammar abstraction is not an appropriate model for how music is processed in the mind. In relation to the *generation* of musical structures described in this chapter, it has already been argued that the goal is a computationally synergistic system, rather than a cognitive model capable of explaining or simulating musical creativity (Section 11.2). As computers have permeated the compositional process for many musicians, the independence of cognitive and purely generative models becomes increasingly outmoded. Computational processes, such as those described here, may form an integral part of the compositional process, with semantics determined by the grammar designer, most often through conscience to feedback relationships with the generative processes. As has been discussed, such processes may often exhibit emergent properties of their own.

11.4.4 Sequential and Parallel Re-writing

In techniques that use Chomsky grammars, the rewriting is *sequential*, that is, symbols are replaced one at a time sequentially across the string (corresponding to the serial nature of the process that the grammar represents). In the system described in this chapter, *parallel* rewriting grammars are adapted for composition. Due to constraints discussed in Section 11.5.2, rewriting is not fully parallel as is traditionally the case with L–systems as defined in Section 5.3.2.1 (upon which the system is based). Rewriting proceeds on parallel sets of elements in the string after the events represented by the string have occurred.

11.5 Grammars for Music Composition

This section details how L-systems may be used to generate musical sequences. It is based on the L-system models and definitions introduced in Chapter 5.

11.5.1 L–Systems

As previously detailed, L-systems when used in computer graphics applications adopt a *turtle interpretation* of the produced string to construct the geometric description of the geometric model.

L-systems provide a compact way of representing complex patterns that have some degree of repetition, self-similarity or developmental complexity. In the case of music generation, a geometric interpretation is unsuitable, so individual symbols are interpreted as events and components of event spaces.

In a similar way that extensions to DOL-systems, such as parametric and stochastic L-systems, have benefited the geometric modelling capabilities in graphics applications, those extensions can also provide greater flexibility in a musical interpretation. For example, the use of a hierarchical specification, detailed in Section 11.5.5, allows a more complex sequence of events and gives the user greater potential for modelling interacting event spaces at different levels.

11.5.2 Music with DOL–Systems

For this section, we assume a D0L-system as defined in Section 5.3.2.1. A simple method of generating music is to map elements from the alphabet V to directly represent musical notes. Symbols can represent notes absolutely (e.g. the symbol *C4* represents middle C), or as ordered pairs $\left(n, r\right)$, representing *note* and *register*. At each iteration, the generated string can then be interpreted as a series of musical events.

For example, given the set of productions and an axiom:

$$\begin{aligned} \omega &: \quad C \\ p_1 &: \quad C \to E \\ p_2 &: \quad E \to CGC \\ p_3 &: \quad G \to \varepsilon \end{aligned} \tag{11.2}$$

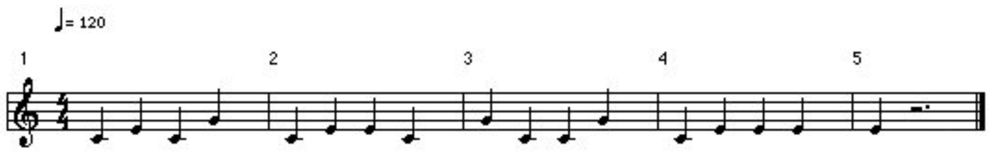
The strings produced for the first five iterations are:

ITERATION	STRING
0	C (axiom)
1	E
2	C G C
3	E E
4	C G C C G C
5	E E E E

Concatenating the strings together gives the sequence:

C E C G C E E C G C C G C E E E E

If each symbol is interpreted as a musical note (at the default register in this example), the string can be played as music:



This simple example has no provision for note timing, so each note defaults to a quarter–note duration. It is of course possible to incorporate symbols in the grammar that control duration.

11.5.3 Stochastic D0L-systems

A stochastic 0L-system, G_π, defined in Section 5.3.4.1, associates a set of probabilities, π, with the set of the productions, *P*. The probability of an individual production being used is usually written above the arrow symbol. For example the production:

$$p_1: \; a \; \begin{matrix} \xrightarrow{1/2} b \\ \xrightarrow{1/2} c \end{matrix}$$

means that a has equal probability of being replaced by b or c. It is assumed that, for all productions with a as the predecessor, that the combined probabilities for that symbol sum to 1, i.e.:

$$\sum_{p_i \in P(a)} \pi(p_i) = 1$$

Stochastic grammars allow the representation of Markov models. For example, the transition matrix shown in Figure 11-1 is equivalent to the stochastic L-system:

$$\begin{aligned}
p_1 &: C \xrightarrow{2/3} D \\
& \xrightarrow{1/3} E \\
p_2 &: D \xrightarrow{3/10} C \\
& \xrightarrow{3/10} D \\
& \xrightarrow{4/10} E \\
p_3 &: E \xrightarrow{5/11} D \\
& \xrightarrow{5/11} E \\
& \xrightarrow{1/11} G \\
p_4 &: G \xrightarrow{1/2} E \\
& \xrightarrow{1/2} G \\
\omega &: E
\end{aligned}$$

The choice of axiom is arbitrary. For a correct simulation of the Markov chain the axiom should select a starting symbol based on the frequency a note appears in the set of input notes (this could be incorporated into the grammar as an additional production).

11.5.4 Parametric Extensions

In the case of parametric L-systems (Section 5.3.5) in a musical application, parameters can be used to control continuous–valued attributes such as note velocity. The MIDI[82] specification allows for a number of continuous–valued controllers. These controllers can be affected through parametric association with the appropriate symbol.

11.5.5 Hierarchical L-systems

As stated, many forms of music involve repeating themes and patterns, often at a number of different levels in a composition. A classic example of this hierarchy being the sonata form, popularised by Joseph Haydn in the eighteenth century. While it is possible to incorporate many levels in a set of pro-

[82] Musical Instrument Digital Interface (MIDI) is a standard method of control in electronic music. It allows the connection of various types of computers, synthesisers and keyboard controllers, provided they obey the MIDI protocol (Rothstein 1992).

ductions, a naturally easier way is to impose a hierarchical containment of rules in order to better represent the structure of the music.

Hierarchical L-systems in the sense used here, have a similar specification as a DOL–System grammar, except that on the successor side of a production, one or more of the successor symbols may be an entire grammar in itself. These are related to sub-L-systems proposed by Hanan (Hanan 1992), although sub-L-systems apply different sets of rules to different portions of the string. They are also related to decompositions (Prusinkiewicz et al. 2001) and the hierarchical model described in Chapter 9. Development of each set of rules proceeds independently, however rule sets may pass parameters between sets of grammars. Figure 11-3 shows an example grammar and the resulting strings produced from the rules.

```
A = {
    B(x) = {
        ω:       a(x)
        p_b1 :   a(y) → a(y+1) c(y/2)
    }
    ω:       B(1)
    p_a1 :   B(x) → a(x) [ B(2x) a(x) ]
}
```

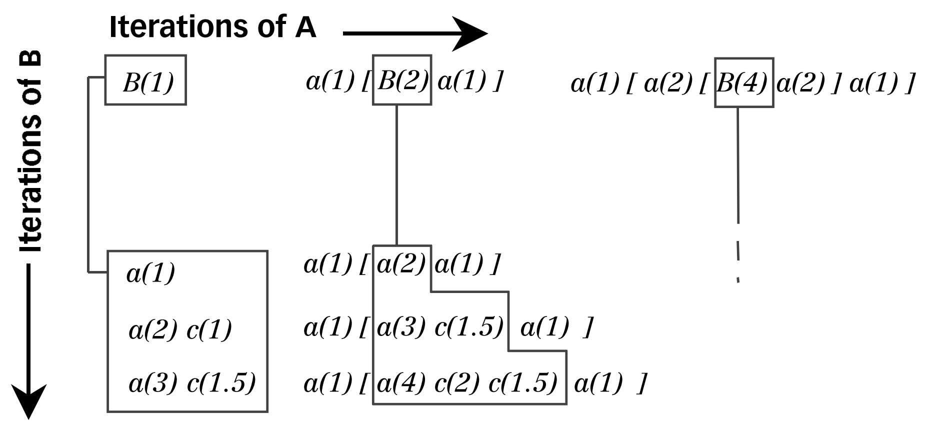


Figure 11-3: A simple hierarchical grammar consisting of two rule sets, **A** and **B**. Below the rules the diagram shows the produced strings, with iterations over system **A** running horizontally and iterations over system **B** running vertically (the iteration of **B** for the third iteration of **A** is not shown). Each system's development proceeds independently, but they may communicate via parameters.

11.6 Implementation

Thus far, various types of L-systems have been introduced and some simple interpretation of the strings they produce for musical purposes has been described. In this section, the implementation of a composition system based on the systems described in the previous sections is discussed.

The system is implemented on a Silicon Graphics workstation connected to a digital synthesiser via MIDI. Once a grammar is parsed, it is capable of generating note sequences (MIDI data) in real–time. Very complex grammars (such as those with many levels of hierarchy, or productions that cause exponential growth in the produced string) may cause noticeable delays in the system during processing. In this case, sequences can be saved onto disk for later playback. The goal of the system is real–time performance and in most cases, this is adequately accommodated.

11.6.1 Data Flow

Essentially, the composition system has three major processing stages, outlined in Figure 11-4. The system generates strings that are interpreted as musical data. Productions are applied to these strings and the results interpreted as commands that are output as MIDI data to various synthesisers connected to the system. MIDI data can be converted to standard notation, suitable for interpretation by real musicians (as opposed to machines) if required.

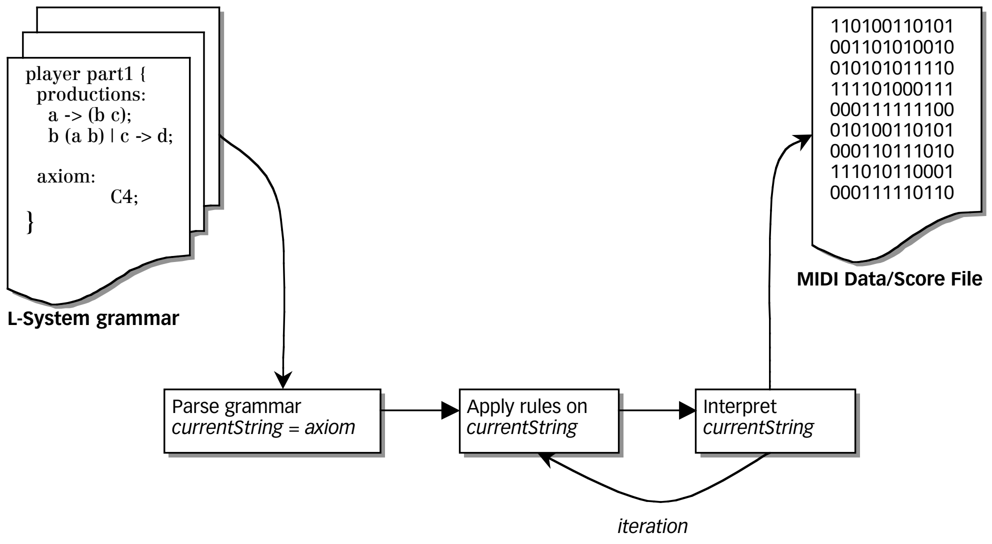


Figure 11-4: Stages of processing in the system. Input and output data are show and the processes that connect them. The lines with arrows show the directional flow of information. First, a grammar is parsed, and the axiom loaded into the variable `currentString`. Rule application is iterative. After rewriting is applied to `currentString`, the results are interpreted. The interpretation converts symbols into MIDI data that is then played, or saved to a file.

11.6.2 Players and Symbol Interpretation

As described in Section 11.5.2, the turtle interpretation used for geometric model generation is unsuitable for music generation. Hence, the turtle interpretation is replaced with the concept of a *virtual player*. Like the turtle, the player is responsible for the interpretation of instructions given to it, which in the general case means playing notes (sending MIDI messages or writing score information) Players always maintain a current *state,* that consists of:

- pitch (note and register);
- duration;
- timbre;
- control parameters.

Control parameters may be used to specify continuous or discrete controllers (e.g. pitch bend, filter resonance, etc.).

Symbols in the produced string change the player's state. The string is read sequentially, from left to right and the player interprets each symbol, just like a Turing machine reads instructions from a tape. Different symbols cause different components of the player to change state, or perform a task. For example, the dot (".") symbol tells the player to play the current note(s), a "+" increments the current note by a semitone, a "–" decreases it by the same amount. Players are capable of playing any number of notes simultaneously, however, in practice this may be limited by output hardware. The square parenthesis symbols ("[" and "]") push and pop the current player's state onto a First–In, Last–Out stack.

Multiple players perform a *composition*. Any number of players may be involved in a composition, however again hardware limitations may fix an upper limit on the total number of notes playable simultaneously. Generally, each player plays a single voice or instrument and each has a different set of rules.

Each virtual player usually has their own set of generating L-systems (although the same L-system can be shared by more than one player, for example when it is desired to have different voicing for the same melody). Each player's L-system generates concurrently, with synchronisation between players achieved by synchronising the derivation steps. It is up to the designer of multiple players' parts to keep them synchronised in terms of time and key signatures. Due to the synchronisation of derivation steps, all players must effectively perform under a *master tempo* which controls the real-time interpretation of derivation strings into MIDI data (this is achieved collectively for all players).

Individual notes can be represented by their name. The "#" symbol represents a sharp, "&" a flat and "@" a natural. Notes in uppercase change the pitch and play the note. Notes in lowercase change the pitch but don't play the note. This means that "C" is equivalent to "c .". A comma "," plays a rest for the current duration. Other symbols change duration, timbre, and so on.

Polyphonic playing of notes is achieved by enclosing the notes to be played simultaneously in round parenthesis: "(" and ")". For example the string:

```
C E G
```

plays the specified notes, each for the current duration, in temporal sequence from left to right. The string:

```
( C E G )
```

plays the same three notes, for the current duration, simultaneously (a C Major chord). Pitch specification can be relative, thus:

```
c ( . + + + . + + + . )
```

Does the same thing (plays a C Major chord). This would not be recognised in the same context as the explicit (C E G) chord, however (see next section).

11.6.3 Polyphony and Context

Context sensitivity relates production application to the predecessor's context (symbols that appear before the predecessor in an axiom or string). In the case of music composition, context sensitivity is represented in the grammar both polyphonically (notes currently being played) and temporally (note(s) previously played). A vertical bar ("|") is used to signify temporal context: symbols to the left of the bar must precede the current symbol for the rule to apply. Table 11–1 illustrates some examples. The interpretation of context relationships is also dependent on the broader issue of construction and interpretation of produced strings to produce musical data. For the interpretive system of Section 11.5.2, derivation strings are concatenated meaning temporal transformations propagate over the entire composition, not necessarily in directly adjacent temporal contexts. Using a timed L-system interpretation or the cellular developmental model of Chapter 9, produces different results as derivation strings are not concatenated. This is further discussed in Section 11.7.2.

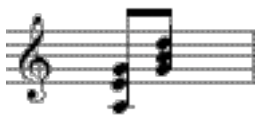
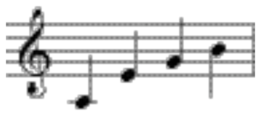
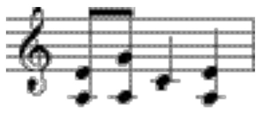
String	Interpretation	Musical Representation
(C E G) → (G B D)	*Polyphonic context:* if current notes are *C, E* and *G* play *G B* and *D* simultaneously.	
C E | G → D	*Temporal context:* if current note is a G preceded by E and C then play a D.	
(C E) | (G C) → D (C E)	*Polyphonic and temporal context:* if current notes are G and C and they were preceded by C and E played simultaneously then play D for the current duration followed by C and E simultaneously.	

Table 11–1: Context sensitive strings and their musical interpretation.

The use of context in a grammar allows complex shifting themes to propagate through a composition, both chromatically and temporally. Context is necessary for a grammar to implement Markov models of order 2 or higher.

11.7 Application of Developmental Systems

Thus far, the symbolic nature of string replacement has been the focus of the system described. Some symbols are designated to represent pitches or pitch changes. In the models described in Section 11.5.2, arbitrary note durations were assigned to such symbols (L-system 11.1) hence no direct control of duration was possible within the grammar.

A number of other systems perform a post-processing step on the produced string, where timing and other data is added (Supper 2001). The process for this stage may be executed by machine or composer. In a similar way that DOL-systems specify topology in a geometric interpretation, in the musical context they specify a *musical topology* — events and the relationship between events. Further interpretation is required in order to produce a completed score.

11.7.1 Specification of Timing Information

A natural extension to allow timing data in the L-system model involves parametric L-systems. A specific parameter can denote a duration, either an absolute time value or as an index to a pre-defined list of durations. Such a list could contain timings for the standard note durations from semibreve to hemidemisemiquaver. Tied or dotted notes could be created with additional symbols in the alphabet that modify the timing interpretation for the number of notes specified by the modifier symbol's parameter. This is illustrated in Figure 11-5.

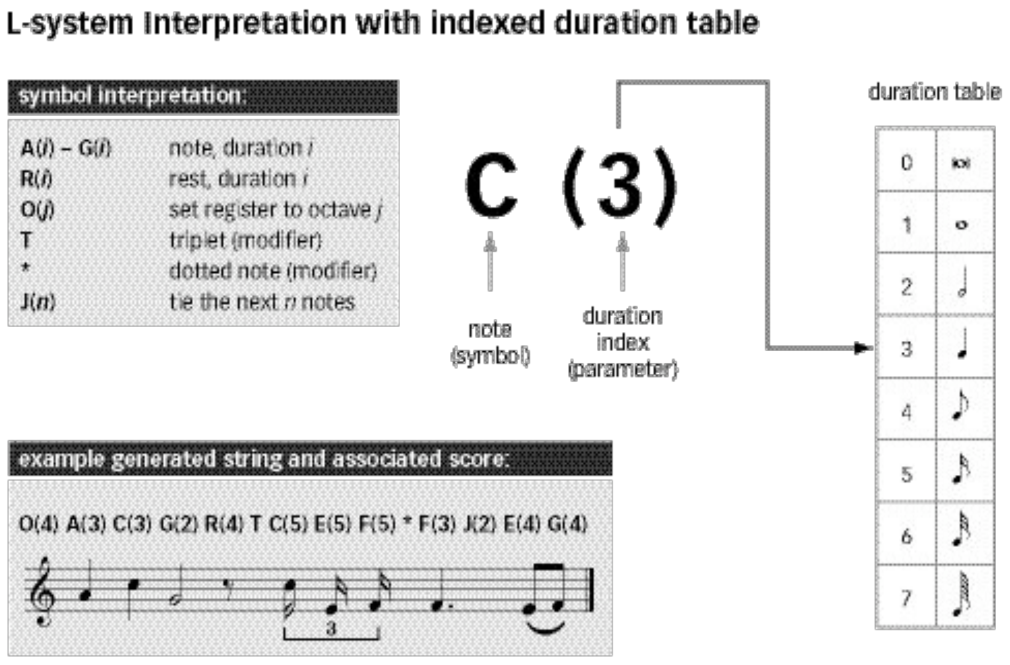

L-system Interpretation with indexed duration table
symbol interpretation:
A(i) – G(i) note, duration i
R(i) rest, duration i
O(j) set register to octave j
T triplet (modifier)
* dotted note (modifier)
J(n) tie the next n notes
C (3)
note (symbol)
duration index (parameter)
duration table
0 1 2 3 4 5 6 7
example generated string and associated score:
O(4) A(3) C(3) G(2) R(4) T C(5) E(5) F(5) * F(3) J(2) E(4) G(4)
3

Figure 11-5: String interpretation using an indexed table and note parameters to specify note-based timing. Modifier symbols in the alphabet enable extra note timing such as dotted and triplet notes.

As with plant models and drawing on the formal properties of L-systems (Frijters & Lindenmayer 1974; Lindenmayer 1974), similar results could be achieved with context sensitive L-systems. This would require replacing the parameters with additional symbols to specify timing information, in most cases resulting in a vast increase in the size of the alphabet. This added complexity would make authoring L-systems to generate compositions more difficult.

The parametric representation with indexed duration, while adequate for basic purposes, has a number of limitations. These include:

- timing data is mixed with other parameters making it more difficult to distinguish the two;
- complex timing is difficult or impossible to achieve, particularly when the composer wishes to differentiate different parameters and their musical context;
- each note event requires an explicit timing parameter, possibly resulting in a large amount of redundancy in the specification, particularly at any level in the hierarchy above the note level.

To address these issues, timed L-systems, introduced in Chapter 8, and the cellular developmental system, introduced in Chapter 9, are a natural extension to apply to the generation of musical data. Timed L-systems build temporal information into the developmental model and so can be used also to generate timing information for music. The cellular developmental model has been designed to enable the modelling of complex, hierarchical time-based systems, again, ideally suited to music generation. The musical interpretation of cells was discussed in Section 9.4. Additional examples are given in the following section.

11.7.2 Timed L-systems and Developmental Systems

Timed L-systems associate an age parameter that continuously increments as development proceeds. This gives individual symbols the ability to carry timing information. The life of the module can then represent the lifetime of an individual note or event. For example, this timed L-system:

$$
\begin{array}{ll}
\omega: & bar \\
p_1: & bar \rightarrow (K,0)(R_S,0)(HC(0),0) \\
p_2: & (K,QN) \rightarrow (R_K,0) \\
p_3: & (R_K,QN) \rightarrow (K,0) \\
p_4: & (S,QN) \rightarrow (R_S,0) \\
p_5: & (R_S,QN) \rightarrow (S,0) \\
p_6: & (HC(n),EN): n \le 7 \rightarrow (HC(n+1),0) \\
p_7: & (HC(n),EN): n > 7 \rightarrow (HO,0) \\
p_8: & (HO,EN) \rightarrow (HC(0),0)
\end{array}
\tag{11.3}
$$

generates a simple 4/4 rhythm. The constants *QN* and *EN* represent quarter- and eight-notes respectively. Instrumentation is represented by the other symbols: *K* represents the kick drum, *S* the snare, *HC* the high-hat closed and *HO* the high-hat open. After one bar, the generated results are shown in the score below. Additional parameters to the drum symbols can be used to control volume, giving the ability to create accents.

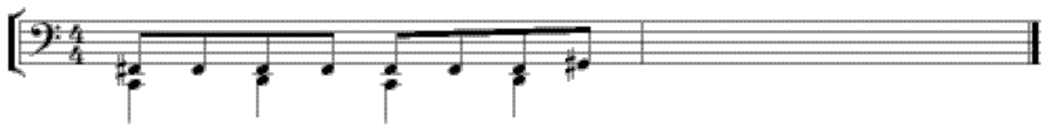


Figure 11-6: Simple rhythm generated using the timed L-system (11.3)

For timed symbols that represent structural rather than musical information, one needs to be careful that the timed derivation of structural symbols does not conflict with the intended timing of the composition. In the example above for instance if the symbol *bar* carried timing information, its timed derivation would delay the actual musical sequence by its lifetime.

The cellular developmental system, described in Chapter 9, gives further expressiveness to musical specification. For example, Figure 11-7 below shows a simple two-chord sequence with both volume and filter control modulated by cell development (please refer to Appendix B for details on the music modules used in the `geom` sections). We assume that these modules are controlled by a parent system that sets the root note for the chords. The use of square brackets (“[” and “]”) works as in the geometric case, saving and restoring musical state (note, register, volume, pan, etc.).

```
module triad() {
   real filterf = 0;
   real volume = 0;
 rules:
   : filterf < 1 : rate filterf = 0.35 * filterf + 0.1;
   : volume < 1 : rate volume = 1/(WN * 4);
   : age > WN * 4 : -> diminished(filterf);
 geom:
   [ vol(volume) cc(FILTERF,filterf) N incN(3) N incN(2) N ];
} // triad

module diminished(real fm) {
   real filterf = fm;
   real volume = 1.0;
 rules:
   : filterf > 1 : rate filterf = -1.35 * filterf;
   : volume < 1 : rate volume = -1/(WN * 2);
   : age > WN * 2 : -> ;
 geom:
   [ vol(volume) cc(FILTERF,filterf) N incN(3,-1) N incN(2,-1) N ];
} // diminished
```

Figure 11-7: Two chord pattern with volume and filter control

Volume and filter control are specified as differential equations with boundary conditions (the variables `volume` and `filterf`). This information is sent using the commands `vol` and `cc` respectively as normalised values. The `cc` command stands for MIDI continuous controller and it is assumed that the `FILTERF` constant represents the appropriate controller value. A synthesizer interpreting the MIDI information output would take the controller value and use it to adjust the cut-off frequency of a low-pass filter, thus modulating the timbre of the sound generated. As the filter frequency increases, so does the volume, the sound building to a crescendo. After four bars (four whole notes, `4*WN`), the chord changes to a diminished version of the triad and recedes in filter frequency and volume, until silence.

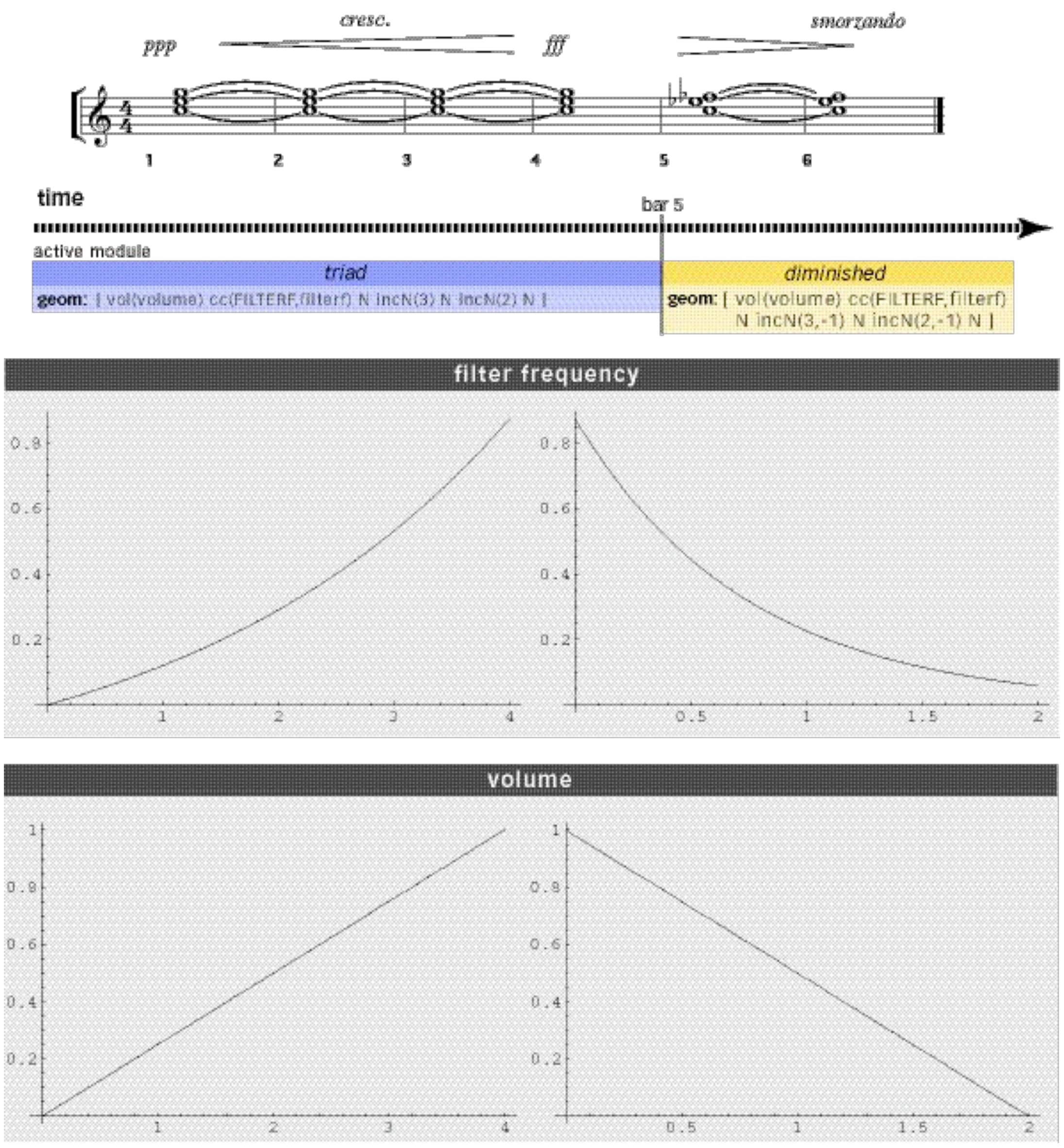


Figure 11-8: The two chord sequence of Figure 11-7 showing plots of continuous changes in volume and filter frequency.

11.7.2.1 Song Structure

The hierarchical structure of the cellular system allows the composer to specify musical semantics as decompositions. Using the cellular developmental model both system cells and module cells can be used to aid in construction of higher-level (non-note playing) compositional structure, with systems useful for structural decomposition, modules for temporal development and sequences such as chord progressions (described in the next section).

A classic musical form is the *sonata form*, which dates from the sixteenth century. The sonata was designed for instrumental music, corresponding to rise in importance of instruments (as opposed to voice) at the time. The classic sonata form is a single movement with three basic divisions: exposition, development, and recapitulation. The exposition and recapitulation sections divisions can be further divided, as shown below in Figure 11-9.

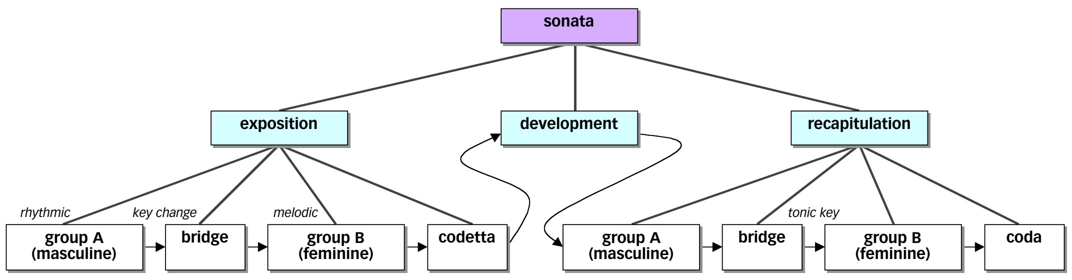


Figure 11-9: Hierarchical decomposition of the sonata form. The lines with arrows show progression through the sonata.

As the figure shows, the exposition and recapitulation divisions are similar, but the recapitulation will usually be a modified version of the exposition, with the second group (B) in the tonic key and the full coda conclusion. The sonata can be represented by the following cellular system (Figure 11-10).

```
system sonata() {
   system exposition() {
      module groupA() {
       rules:
         : age > EGA : -> bridge(MINOR);
      }
      module bridge(integer k) {
       rules:
         : age > EB : -> groupB();
       geom:
         key(k);
      }
      module groupB() {
       rules:
         : age > EGB : -> codetta();
      }
      module codetta() {
       rules:
         : age > EC : -> sonata::development();
      }
   } groupA();

   system development {
      module developmentSequence() {
       rules:
         : age > DDS : -> sonata::recapitulation();
      }
   } developmentSequence();

   system recapitulation() {
      module groupA() {
       rules:
         age > RGA : -> bridge(TONIC);
      }
      module bridge(integer k) {
       rules:
         : age > RB : -> groupB();
       geom:
         key(k);
      }
      module groupB() {
       rules:
         : age > RGB : -> coda();
      }
      module coda() {
       rules:
         : age > RC : -> ;
      }
   } groupA();
// axiom for the sonata
} exposition();
```

Figure 11-10: The sonata form encoded as a cellular developmental system.

For space and clarity, the figure leaves out the actual musical modules — these would be included in a full realisation of the work. The uppercase constants represent timing information, with the exception of the `TONIC` and `MINOR` constants, representing the appropriate key shifts. Notice also, the use of scope to instantiate modules or systems outside of the current nesting (e.g. the call to `sonata::development` from within the module `codetta`. The hierarchical cell specification makes it easy to "decompose" a composition into structural and temporal units, allowing easy modification of individual components within the structure. This makes for a more flexible and robust environment than a more simple flat grammar would provide.

11.7.2.2 Context and Chord Progressions

This example illustrates how chord progressions can be generated. Traditional musical language specifies harmonies that follow certain logical orders based on acoustic, aesthetic, and psychological experience (Karolyi 1965, pp. 67-68). In the example below the collection of modules generates chord progressions based on harmonic rules.

```
module tonic(real d) {
   integer n = random(1,7);
   integer rshift = random(-1,1);
 rules:
   om(d, r) me: age > d && n == 1 && r > 1: -> om(d,-1) me(d);
   om(d, r) me: age > d && n == 1 && r < 1: -> om(d,1) me(d);
   om(d, r) me: age > d && n == 1 && r == 0: -> om(d,rshift) me(d);
   : age > d && n == 2 : -> supertonic(d/2);
   : age > d && n == 3 : -> mediant(d*2);
   : age > d && n == 4 : -> subdominant(d*2);
   : age > d && n == 5 : -> om(d*2,0) dominant(d*2);
   : age > d && n == 6 : -> submediant(d/2);
   : age > d && n == 7 : -> seventh(d/2);
 geom:
   key(1);
} // tonic

module suptertonic(real d) {
   integer n = random(1,5);
 rules:
   : age > d && n == 1 : -> om(d/2,1) dominant(d/2);
   : age > d && n == 2 : -> mediant(d);
   : age > d && n == 3 : -> subdominant(d);
   : age > d && n == 4 : -> submediant(d);
   : age > d && n == 5 : -> seventh(d/2);
 geom:
   key(2);
} // supertonic
```

```
module mediant(real d) {
   integer n = random(1,4);
 rules:
   : age > d && n == 1 : -> submediant(d);
   : age > d && n == 2 : -> subdominant(d);
   : age > d && n == 3 : -> supertonic(d);
   : age > d && n == 4 : -> om(d/2,1) dominant(d/2);
 geom:
   key(3);
} // mediant

module subdominant(real d) {
   integer n = random(1,6);
 rules:
   : age > d && n == 1 : -> om(d/2,-1) dominant(d/2);
   : age > d && n == 2 : -> tonic(d);
   : age > d && n == 3 : -> submediant(d);
   : age > d && n == 4 : -> supertonic(d);
   : age > d && n == 5 : -> seventh(d/2);
   : age > d && n == 6 : -> mediant(d);
 geom:
   key(4);
} // subdominant

module dominant(real d) {
   integer n = random(1,4);
 rules:
   om(d,r) : age > d && n == 1 && r > 0: -> om(d/2,-1) tonic(d/2);
   om(d,r) : age > d && n == 1 && r < 0: -> om(d/2,1) tonic(d/2);
   om(d,r) : age > d && n == 1: -> om(d/2,r) tonic(d/2);
   : age > d && n == 2 : -> submediant(d);
   : age > d && n == 3 : -> mediant(d);
   : age > d && n == 4 : -> subdominant(d);
 geom:
   key(5);
} // dominant

module submediant(real d) {
   integer n = random(1,4);
 rules:
   : age > d && n == 1 : -> supertonic(d);
   : age > d && n == 2 : -> om(d*2,-1) dominant(d*2);
   : age > d && n == 3 : -> subdominant(d);
   : age > d && n == 4 : -> mediant(d);
 geom:
   key(6);
} // submediant

module seventh(real d) {
   integer n = random(1,4);
 rules:
   : age > d && n == 1 : -> tonic(d*2);
   : age > d && n == 2 : -> submediant(d);
   : age > d && n == 3 : -> mediant(d);
   : age > d && n == 4 : -> om(d*2,-1) dominant(d*2);
 geom:
   key(7);
} // seventh

module om(real d, integer r) {
   // this module sets the chord register
 rules:
   age > d -> ;
 geom:
   incR(r); // sets relative register
} // om
```

Figure 11-11: Harmonic chord progression modules.

Each of the main modules represents a chord (I, II, III, IV, V, VI, VII). The module `om` is used a register marker and the temporal context of the main tonic and dominant chords take this marker into account when deciding how to switch registers. This results in a register shifting sequence modulated by the tonic and dominant chords. The `d` parameter controls duration of each module and a simple time shifting method favours longer durations for the tonic, mediant and dominant (I, III, and IV) chords. A sample sequence of chords output from this collection of modules is shown below, assuming the system instigating the sequence begins with the tonic as its axiom.

```
tonic(4) → mediant(8) → om(4,1) dominant(4) → submediant(4) →
supertonic(4) → seventh(2) → om(4,-1) dominant(4) → om(2,1)
tonic(2) → submediant(1) → supertonic(1) → submediant(1) →...
```

Notice how chords progress in timing, register and the chord itself. Selection of one chord from the next involves an element of chance (a selection from the allowable next chord given the current one). In this way, the system resembles a kind of Markov model. The chord modules themselves do not play any notes, they just set the key based on chord. This enables other modules to sequence actual musical data in that key. In fact, these chord changes can serve as the guiding harmony for the composition. As a simple example, the modules listed below for a basic up/down arpeggio.

```
module arpeggio_u(integer n) {
 rules:
   : age > SN && n <= 6 : arpeggio_u(n+1);
   : age > SN && n > 6 : arpeggio_d(n-1);
 geom:
   [N incN(n)];
} // arpeggio_u

module arpeggio_d(integer n) {
 rules:
   : age > SN && n > 0 : arpeggio_d(n-1);
   : age > SN && n <= 0 : arpeggio_u(n+1);
 geom:
   [N incN(n)];
} // arpeggio_d
```

Figure 11-12: Up and down arpeggio modules.

Placing the `arpeggio_u` module after one of the chord shifting modules described in Figure 11-11 creates a sequence of chord-shifted arpeggios. The figure below shows how such a system would work. The system would contain definitions for the modules described in Figure 11-11 and Figure 11-12.

```
system shifter(real d, integer startN) {
   // module defs here
} absN(startN) tonic(d) arpeggio_u(1);
```

Figure 11-13: System for harmonic shifting arpeggios.

The `tonic` and `arpeggio_u` modules develop in parallel, tonic controlling chord shifts and `arpeggio_u` generating arpeggios. There is no specific timing integration between the two activities, in this case ensuring proper time quantisation is important. A more complex system could use additional timing modules to communicate timing information between chord changes and the arpeggios. Additional modules could be added to control (for example) bass and melody, their key changes driven by the chord progression modules that modify the key of the other components, keeping them in all in harmony.

11.7.2.3 Discussion

The examples shown in the preceding sections demonstrate the flexibility and scope of the developmental systems introduced in this thesis. Previous methods were mainly confined to simple discrete systems that specified single note information only. The examples described here show how features such as continuous modulation control; hierarchical decomposition and temporal sequencing; note timing; timbre modification; polyphonic sequencing; and composition dynamics can be incorporated within a unified developmental model. Of course, such a model is more complex than a basic DOL-system specification, for example, meaning a longer learning curve for the composer. However, the additional flexibility gives far greater control and musical possibilities, expanding the creative palette significantly.

11.8 Conclusions and Future Work

This chapter has demonstrated how string re-writing and developmental systems can be applied as a formalised method for the generation of music. The use of stochastic grammars allows the compact representation of Markov chains. The parallel rewriting technique of L–systems has capabilities equal to, or better than, those of previous grammar based methods. Parametric numerical parameters allow the control of non–discrete processes such as attack velocity or volume. Hierarchical and context sensitive grammars allow the simultaneous development of complex patterns at different levels within a composition. The developmental system, introduced in Chapter 9, has been applied to the generation of time-based musical events and event spaces, with a greater degree of timing control over previous discrete grammar-based methods used for generative music. These extended models have addressed many of the criticisms of grammar-based methods (detailed in Section 11.4.3).

In the introduction, it was emphasised that this is a computer assisted composition system, where composer and machine work in a synergetic tandem. Currently this tandem is little better than the way a programmer edits and debugs source code. Grammars must be written, compiled (removing syntax errors in the process) before music can be heard. Currently work is underway to make the system much more interactive from a performance point of view. A composer will author a grammar, and within the grammar place areas of discrete and continuous control that can be influenced by external processes or systems. One such example would be the recognition of hand gestures (gestures have both continuous and discrete information), in the spirit of the GROOVE system of Mathews and Moore (Mathews & Moore 1970). In this way the composer acts as conductor, instantiating medium–term changes in the performance through gesture (discrete gestures cause rule changes) and influencing the short-term quality of the performance (continuous data from hand positions in space). Development of such a system is continuing.

It is also possible to apply the aesthetic evolutionary processes of the previous chapter to evolve better sounding compositions interactively. Starting with a base L-system, a number of different mutated versions of the grammar can be represented as pictorial icons on the computer screen. By moving the mouse towards an icon, the contribution to the composition is increasingly biased towards the associated genotype. If the mouse is completely on the icon, only that phenotype is heard. Once the user finds the best spatial position (and thus the best sounding composition, the phenotype) the selected grammars (genotypes) are "mated" and become the new parent grammar. The process is repeated for as long as the user desires, with the hope of evolving the grammar into a better sounding composition.

While this technique sounds appealing, there is a limit to the number of mutations that can be displayed on the screen at any one time. The number of generations that can be critically evaluated by a human during any one sitting is limited in relation to population sizes traditionally employed by genetic algorithms. We are still a long way from allowing the machine to exhibit creative judgement on its own.

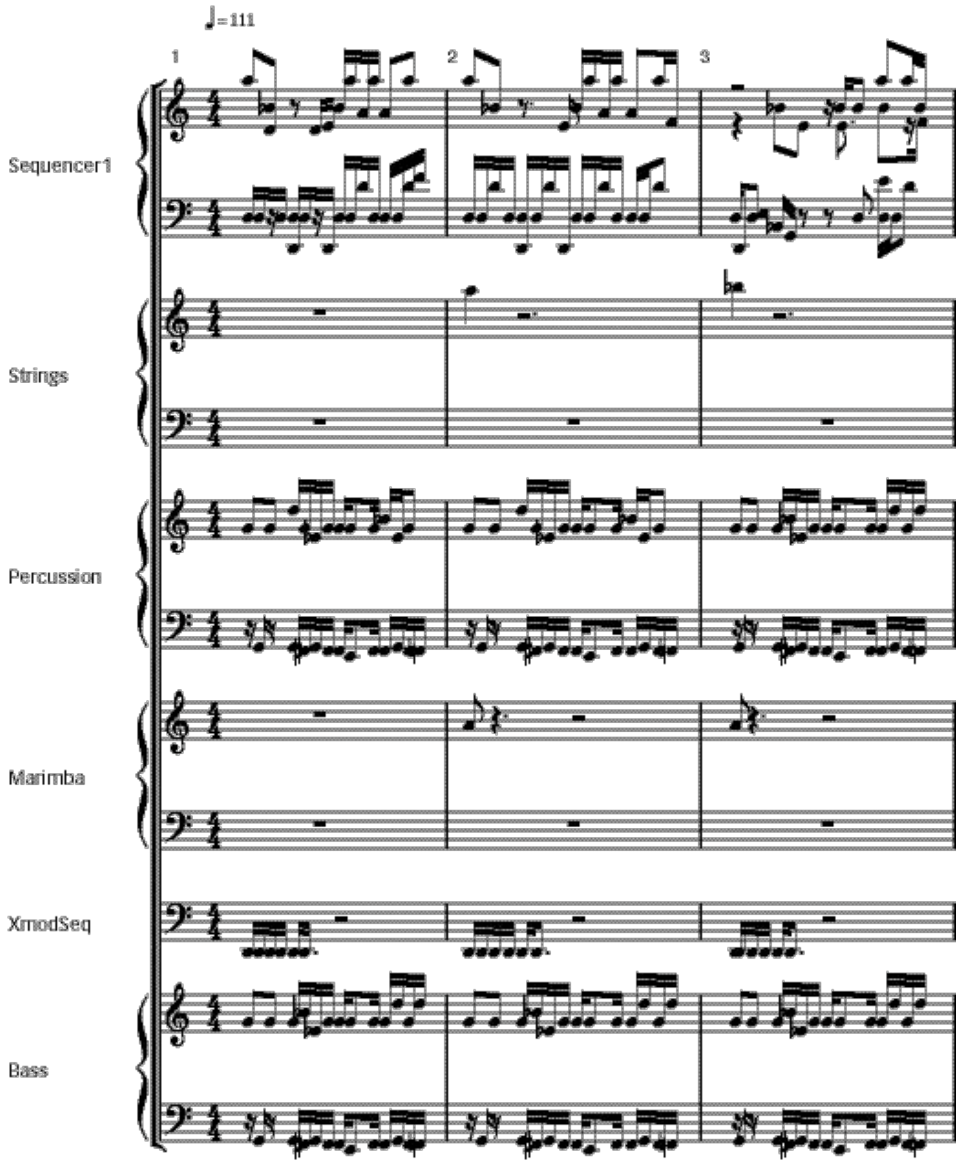


Figure 11-14: Excerpt from a musical score generated using the techniques described in this chapter.

12 Conclusions

Only something which has a purpose comes to an end, since once that purpose is achieved, all that remains is for it to disappear. The human species has survived only because it has no final purpose. Those who have tried to give it one have generally sent it hurtling to its destruction. And it is perhaps out of some survival instinct that groups and individuals are gradually abandoning any precise purpose, abandoning meaning, reason and the Enlightenment to retain only the untutored, intuitive understanding of an imprecise situation.

— *Jean Baudrillard (Baudrillard 2003)*

12.1 Summary of Results

This thesis has covered a wide variety of territory to illustrate the application and possibilities for L-system based models in computer graphics, art, animation and music synthesis.

In summary, my goals in describing the research in this thesis have been:

- the development of a generalised generative system driven by the practical goal of producing new kinds of creative visual and musical works that autonomously change and develop over time;
- research into the concept of emergence and how emergence relates to generative systems, particularly those concerned with novelty and creativity;
- location of the kinds of creative works produced with the generative systems described in this thesis, in terms of a general approach by ar-

tists in representing nature as aesthetic object and as generative system.

12.1.1 Artworks Produced

The software developed and informed by the research detailed in this thesis has been used of the last ten years to produce a number of successful artworks. The most significant of these is the interactive videodisc work *Turbulence: an interactive museum of unnatural history* (McCormack 1994b). First shown at the ACM SIGGRAPH conference in 1994, the work has had more than 40 international exhibitions since this time. Turbulence has also received a number of awards for interactive art, animation and multimedia, including first prize at *Images du Futur* (Canada), winner of the *New Voices/New Visions* award for interactive media (USA), Alias/Wavefront award for computer animation (Art Category, USA) and *Ars Electronica* awards (Interactive art and computer animation categories, Austria). I list these achievements not to grandstand about my artwork, but as (albeit unscientific) evidence that in artistic terms the results of my research could be considered successful.

12.1.2 Topics Addressed in this Thesis

Part 1 examined the contextual history and discourses for using scientific developments, nature and natural systems as a basis for developing creative processes and artworks. I sought to define generative art and its conceptual underpinnings as a unique mode of artistic expression. Historical examples were given that illustrate that the concept of a generative system predates and extends beyond digital instances. Salient issues involving realism, simulation, botanical art, and emergence were discussed. These discussions aid in a contextual and critical analysis of generative art and the technical processes one can exploit in developing such art.

Particular emphasis was given to the concept of emergence and how different categories of emergence can be used to aid critical, developmental, and technical thinking about generative art and the creative process. The concept of the computational sublime was presented as a new category of the sublime made possible by computational processes.

Part 2 described technical research into L-systems and their application to creative synthesis. The turtle interpretation of L-systems was extended to accommodate a rich variety of geometric modelling, with particular emphasis on the use of generalised cylinders, motivated by biological research into the functional morphology of organisms. Further extensions have allowed the production of synthetic landscapes with detailed vegetation, and a good degree of user control. The use of stochastic basis functions, traditionally applied to texturing, was adapted for use in generative modelling. The results illustrated the ability of relatively simple L-systems to model detailed natural scenes with intricate variation.

A key area addressed in this thesis has also been the emphasis on continuous temporal development as an integral component of the formalisms presented here: specifically the timed L-systems of Chapter 8 and the cellular developmental model of Chapter 9. Both these systems allow elaborate and complex animations to be produced, as evidenced in the sequences produced for *Turbulence*. The cellular developmental model combines a number of variations in L-system development into a single system capable of simulating a variety of common generative techniques in a hierarchical specification. The system also combines discrete, continuous and event driven models in a unified system.

The emphasis on interaction and performance lead to an investigation of synchronisation and real-time generation methodologies and algorithms for developmental systems. Real-time development and control of an L-system is important in applications such as music generation and interactive media, where development may be influenced by real-time input from a human user. The synchronisation mechanism presented in Section 9.6.1 gives the benefits of real-time and external synchronisation driven development, along with the increased ability to make maximum use of CPU resources for off-line and software-based rendering applications.

Methods for the aesthetic exploration of the vast space offered by L-system models were presented in Chapter 10. Aesthetic selection was also used extensively for the producing the novel geometric models in *Turbulence*, and in a number of the figures and colour plates in this thesis.

The interpretation of produced strings and the specification of L-system grammars was extended to include the generation of music. New interpretations of context and the use of continuous development to control parameters exceed any of the previous applications of L-systems in this area.

12.2 Computation, Aesthetics and Generative Systems

Computation represents a vast space of possibilities. What remains illusive is any kind of detailed map of this space, and any practical systematic way of exploring it. In a digital computer, patterns of bits (binary digits) are the basic unit of representation. One pattern being different from another is how representations are distinguished. The programmer or hardware designer then "assigns meaning" to these patterns when developing a program or computer. Some assignments follow strict rules that mirror the behaviour or the abstraction to which the same or similar meaning is assigned in other domains. For example, a finite set of integers can be represented by a fixed number of bits. The operation of adding two integers works according to the rules of mathematics (provided one does not exceed the limits of the representation).

Further, a distinction can be made between *iconic* and *symbolic* representation. An iconic sign is one where there is a "topological similarity between the signifier (the sign) and its denotata (what it represents)", whereas a symbol links signifier and denotata by convention only, "without either similarity or contiguity" between the two (Sebeok 1975). Representation assigned to patterns is an activity performed and interpreted by *people*. In theory, a symbolic pattern can represent anything — your income tax, love, a leg — but the representation says nothing about the quality or accuracy of the mapping. The way these representations are made, and how their mappings are devised, is a reflection of our cognitive and perceptual underpinnings.

One possibility in exploring computational space is to minimise representation. A minimal representation, for example could be the presence or absence of a mark (such as a unary representation used in a Universal Turing machine). Similarly, the states of a cellular automaton can be represented as a mark or absence of a mark. The idea being that by minimising the representation, the viewer of the work is free to focus on the phenomenological experience of the process being enacted rather than interpretation of the symbols. One possible approach to the analysis of generative art programs could be to use this goal of minimizing representation as a basis for investigation. Hence, an analytic approach to the complete exploration of computation as a medium might be to first consider an idealized form of computer, such as a UTM (Universal Turing Machine) and systematically examine *every* possible program for such a machine (programs can be represented by marks on a tape and unary coding is sufficient). It might then be possible to classify or categorize many different types of program and hence gain insight into the nature of computation in general, and specific types of programs in particular, such as "the class of programs that perform generative computation" or "those programs that halt". Indeed, this is, somewhat implicitly, one of the goals of computer science.

However, this approach presents a number of difficulties. The search space a simple analysis reveals that the number of possible programs[83], even on the most basic of practical computers, is *vast* beyond imagination. So vast in fact, that any practical systematic analysis would take many more orders of magnitude of time than the universe is old. Such a vastness invokes qualities of the Kantian sublime, in the sense that we have in the computer a programmatic "space" of magnitude beyond comprehension, beyond even the limits of the practical universe in terms of systematic exploration.[84]

Further difficulties arise specifically to the artist, since much of the history of art concerns the issue of representation. In addition, minimisation of a representation is not removal, so the basic problem remains. Without representation, interpretation is difficult, hence when using the computer all representations must necessarily be *symbols subject to interpretation*.

Nonetheless, my approach to this art practice is largely a positivist one. In developing generative software, my goal has been to consider the sensation of generative systems as the basis of phenomenological experience. It is through an interactive engagement with systems that we form our representations about them. By expanding the domain of, and possibilities for these sensations, we expand the "productive psyche" in the terminology of (Bachelard

[83] "Possible program" means *any* program; regardless of whether it stops or does anything 'useful', even in computational terms. In an idealization like a UTM there are no "incorrect" programs.

[84] It might be possible to fully explore our own planet, yet to fully explore computation, in the sense of systematically running every possible program, is currently believed to be impossible.

1958). The software systems I have created are directed towards this goal. I do not see such systems necessarily as mirrors of nature or natural systems, and so they are not always designed as scientific models that represent or simulate known physical phenomena. Hence, they often do not produce the same results as scientific models. This is often the case with the visual simulations found in computer graphics research.

In writing about seventeenth century Dutch realist painting, Ernst Gombrich titled his discussion "The Mirror of Nature". But he was also keen to point out that while Dutch realist art had "learned to reproduce nature as faithfully as a mirror", he also recognised that "Neither art nor nature is ever as smooth and cold as glass. Nature reflected in art always reflects the artist's own mind, his predictions, his enjoyments and therefore his moods." (Gombrich 1984, page 338). In the case of the generative systems described, the processes open the expression of a new nature and hence, they expand our experience of the natural.

Thus far, the most common methodology for this expansion is to have conscious goals when writing a program. These goals determine the code and to some extent, its outcomes. I have discussed the exploration of large aesthetic spaces using biologically inspired search techniques such as aesthetic selection. However, even in these "open" methodologies there is a teleological goal. Nature is non-teleological, one thing such investigations have yet to exploit. Any such search is clouded by the issue of representation, which remains an open problem, spanning a number of disciplines.

Today, art has many different goals, discourses, and agendas. Generative process in art, hence generative art itself, has been a mode of artistic investigation that pre-dates the digital computer. It sets historical precedents for current discussions, precedents that have not received the attention in art theory that they perhaps deserve. In practical terms, the computer offers an extraordinary tool for developing generative processes, the significance of which we are only just beginning to grasp. I hope that the work presented here gives a glimpse into that significance. As now the nature of computation is changing, so too is our experience of these systems, mirrored in this change. However, generative art cannot fall victim to being the "handmaiden of science", so generative artworks of significance must offer insights and experiences beyond a scientistic fascination with generative processes. It is here, in the world of phenomenology and representation, that the possibilities for the generative artist really do remain endless.

12.3 Future Work

The quest for richer and more complex modelling of natural phenomena continues unabated.

One key aspect of the modelling system described in this thesis that forms the basis for future work is the tighter integration of environmental modelling into the developmental model. For example in the cellular developmental model of Chapter 9, cell division, replication, growth, etc. happens without regard for any physical or biological simulation of an environment under which the processes are occurring. Some basic attempts at including environmental input have been included; such as the environmental turtle commands discussed in Section 6.8.3 and the incorporation of tropisms when modelling plants. A particular problem with incorporating more advanced environmental relationships into the model is the problem of how they translate in non-geometric interpretations such as music generation. For these developments, it seems the development must split, sacrificing generality for domain-driven extensions based on specific physical assumptions.

Today, more than thirty years since the original formalism developed by Lindenmayer, L-systems have found application to a wide variety of domains, far beyond the cellular developmental simulation for which they were originally devised. The quality and complexity of realistic plant images generated using L-systems has made them immensely popular in computer graphics applications, largely due to the work of Prusinkiewicz, his colleagues and students.

In recent years L-systems have found more diverse applications including the procedural modelling of cities (Parish & Müller 2001), modelling of biological organs (Durikovic, Kaneda & Yamashita 1998), even in data compression (Nevill-Manning & Witten 1997). This is in addition to the applications that form the focus of this thesis — computer art, animation and music synthesis.

It is interesting to observe that in biology however, there remain a number of sceptics on the explanatory power of L-system models and simulations in explaining *how* the complexity of plant growth occurs at a developmental level (Ball 2001, Chapter 5). Philip Ball points to the attempts of Wilhelm Roux, who at the end of the nineteenth century attempted to specify a series of rules governing branching growth based on empirical studies of arterial networks. These rules were further qualified and extended to trees by Cecil Murray who further applied them to arterial networks and then to trees, showing the power function relation between circumference and weight of a branch distal to a given point (Murray 1927). Much recent research draws on these physiological descriptions, and uses models with cost minimisation criteria (such as surface area, volume and total power losses) as explanatory models for the branching structure observed in plants (Zhi, Ming & Qi-Xing 2001). It is likely that an interesting area of future work would be to combine these cost minimisation constraints with some advanced L-system description to further illuminate the explanation of why particular branching structures and patterns are observed.

A similar situation exists with the phyllotaxis models discussed in Chapter 7. Here again there are two different approaches to the problem of developing a general model of phyllotaxis — the analytical mathematical approach such as that adopted in this thesis where the basic architecture is taken as an empirical given and a model which creates that architecture specified. This certainly gives good and efficient geometric models, but it does lack explanatory power in terms of *why* such patterns form. Here the developmental, causal and physical models provide better explanatory information (Douady & Couder 1992).

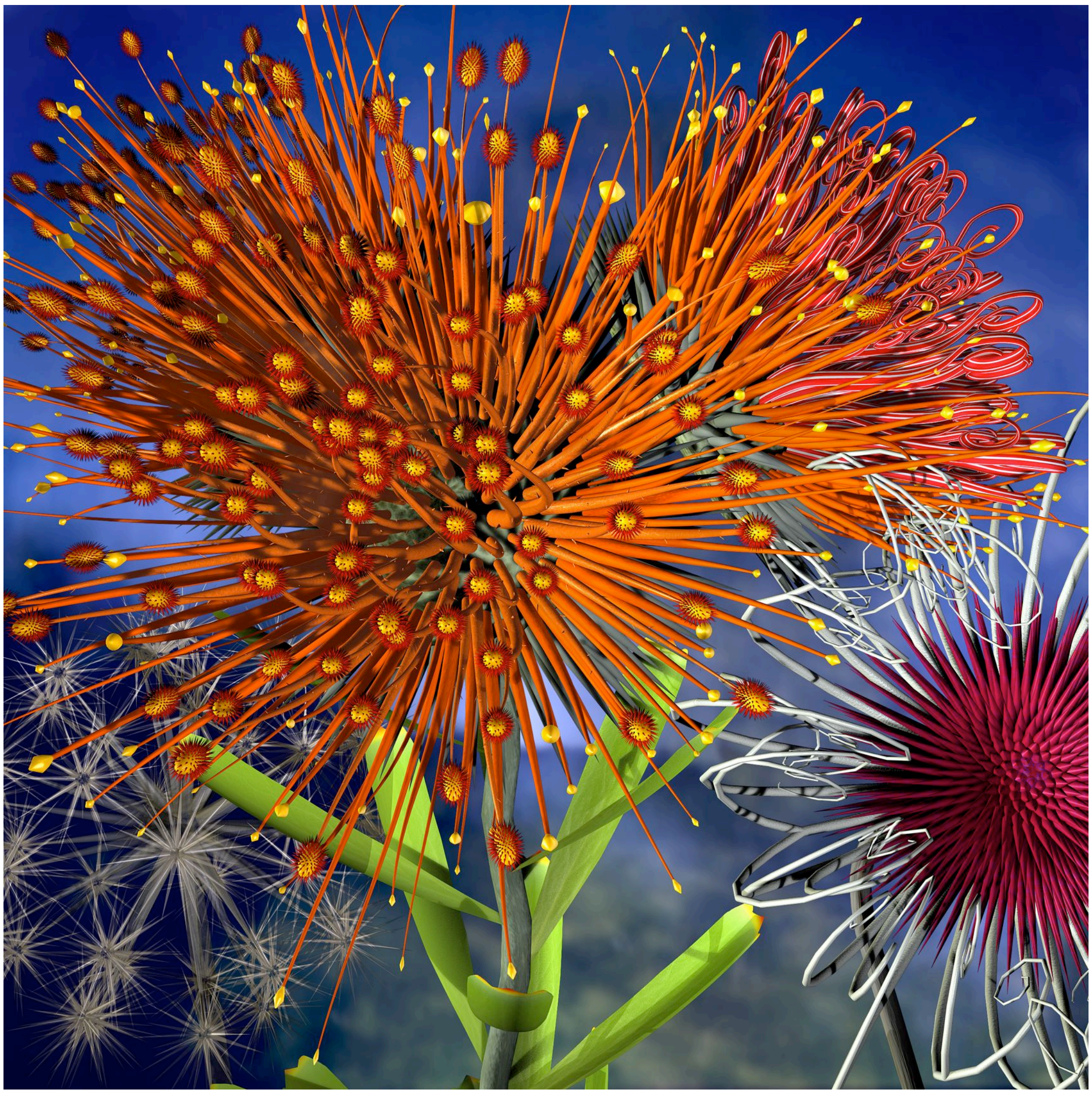
It is the author's opinion that both these methodologies have something to offer, and that they are not opposed, but complementary. Further research may better integrate a variety of approaches, to achieve models that are rich in both descriptive and explanatory modes — a combination of historical and poetic truth, which may lend to the appreciation of a "deeper and richer sense of life".

Every work of art aims at showing us life and things as they are in truth, but cannot be directly discerned by everyone through the mist of subjective and objective contingencies. Art takes away the mist. (Schopenhauer 1928)

Appendix A

Colour Plates

This section contains a number of colour plates. All images were produced using the techniques described in this thesis. Some figures appeared in the main body of the text. In this case, the caption matches the figure as it appears in the main text.


Model created using the interactive evolution techniques described in Chapter 10.

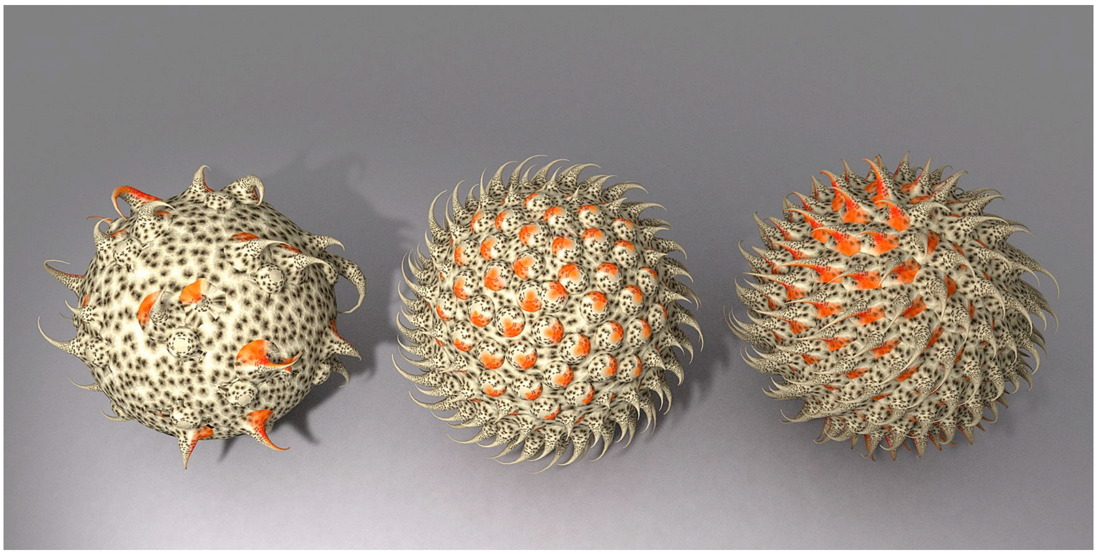


Figure 7-3.

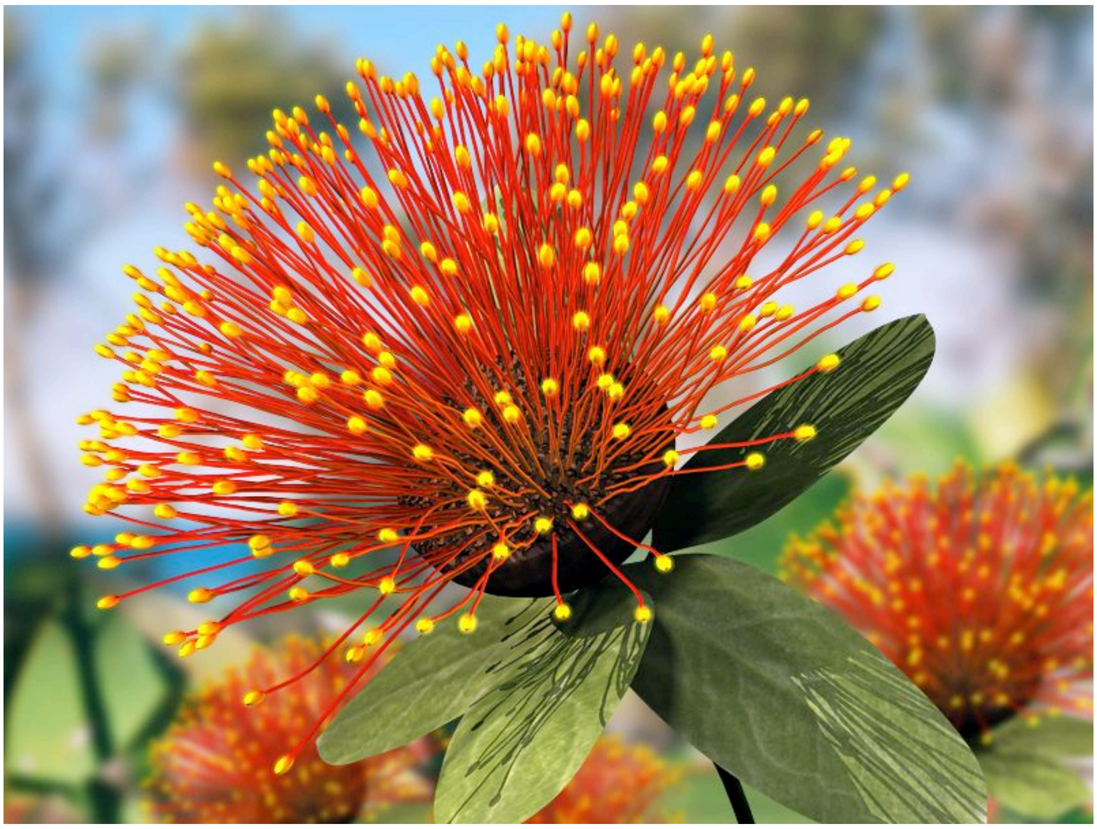


Figure 7-13.

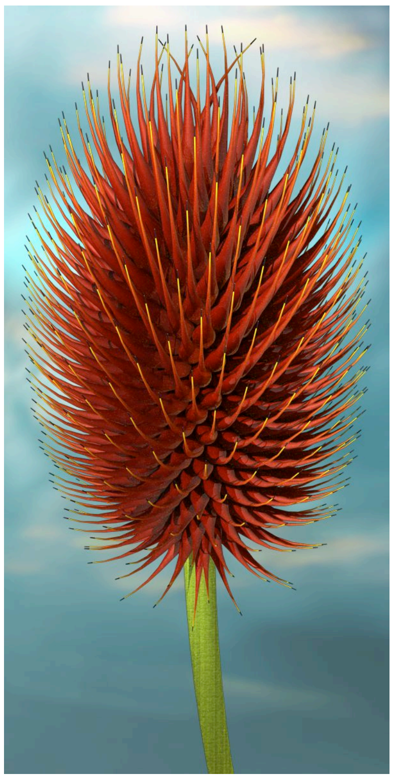
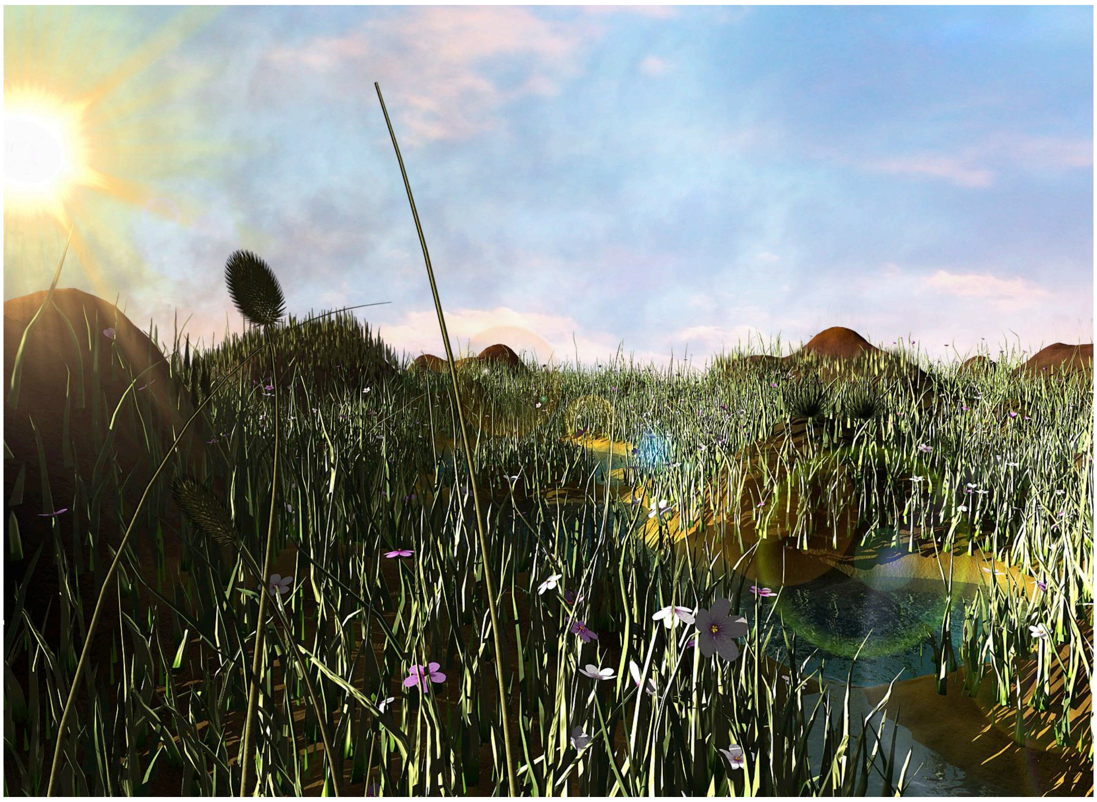


Figure 7-14.

Figure 7-22.



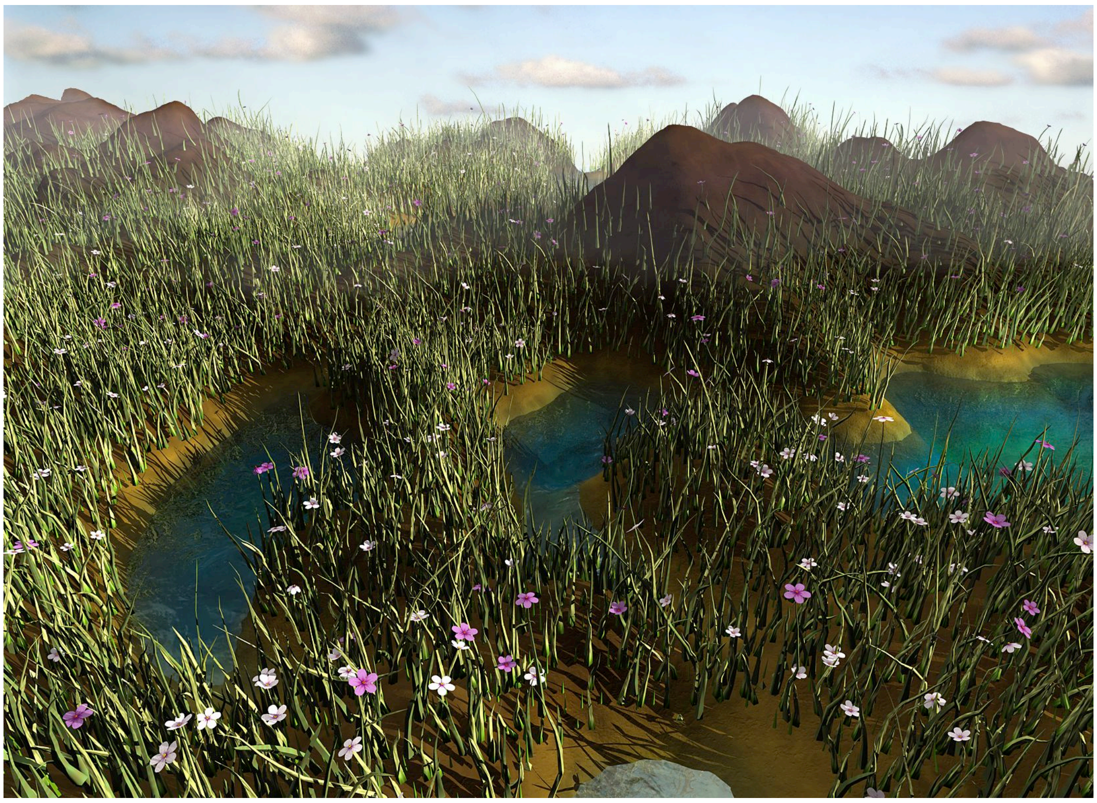
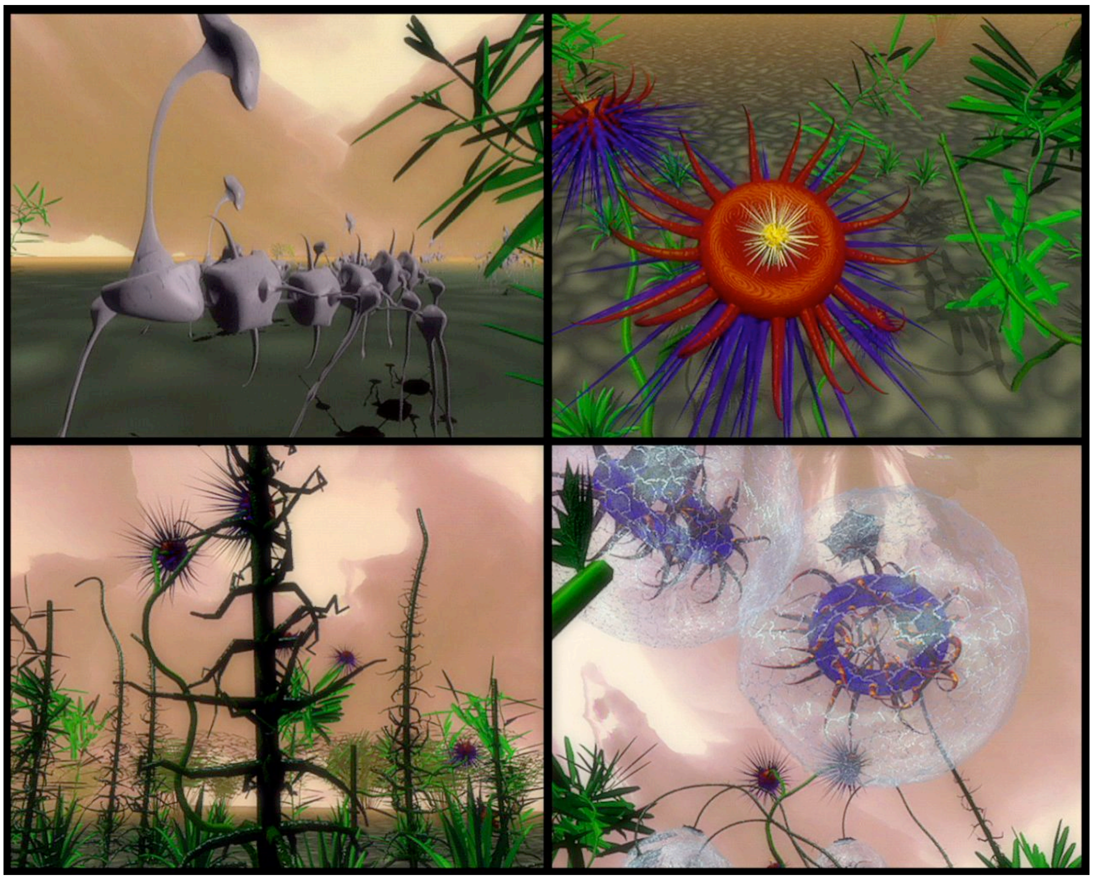


Figure 7-24.


Frames from *Turbulence*.

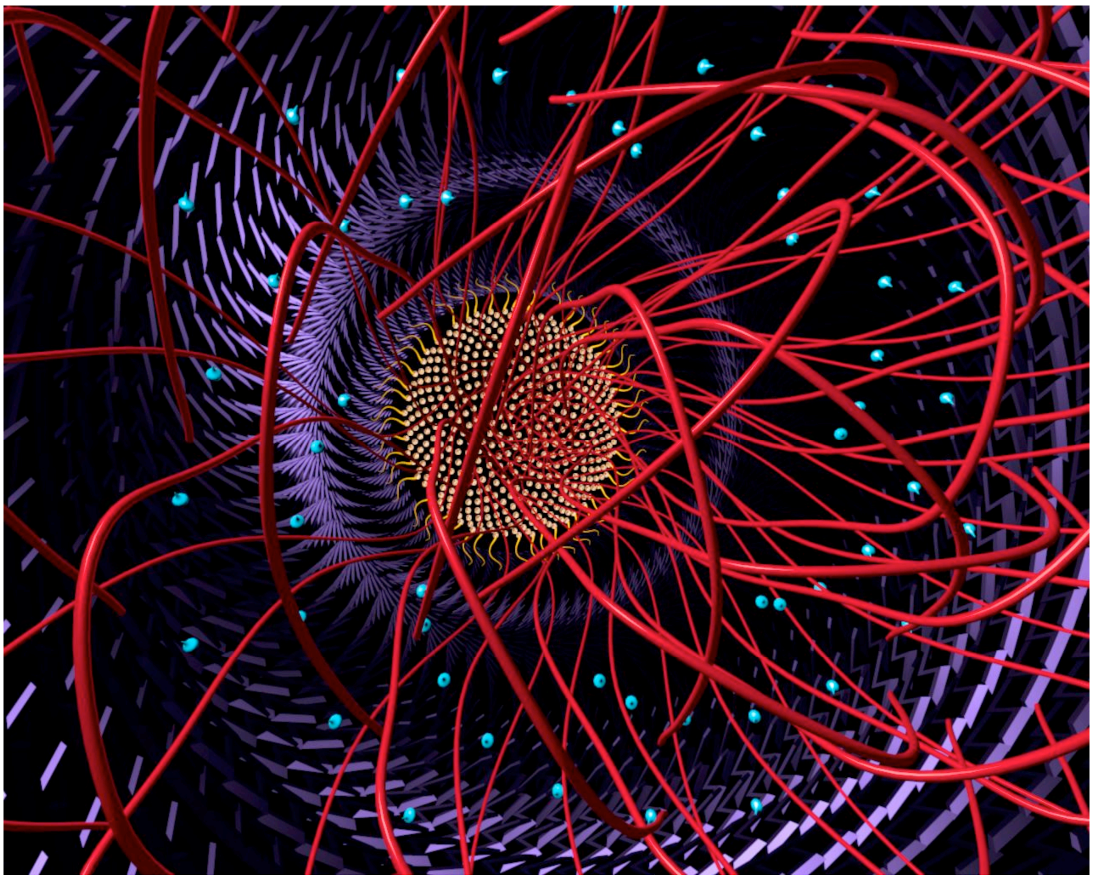


Figure 10-9 (detail).

Image from "TURBULENCE"
© 1995 Jon McCormack

Appendix B

Music Commands

The table below details the music commands used by the cellular developmental model described in Chapter 9. Parameters are optional and if not supplied assume the defaults listed.

Music generation module commands:

Module	*Defaults*	*Description*
`absN(x)`	`x` = 1	Set the current note to *x*, where *x* corresponds to the absolute pitch (the current key).
`incN(x)`	`x` = 1	Change the current note by *x* semitones (*x* may be positive or negative)
`incN(x,n)`	`x=1`, `n` = 0	Change the current note by *x* in the current key. The *n* parameter controls augmented (1), natural (0), or diminished (-1). For example `incN(3,0)` increments by a third, `incN(3,-1)` by a diminished third.
`N(v,rv)`	Last `velN` and `velR`	Play the current note with attack velocity *v* and release velocity *rv*. These are normalised values with 0 corresponding to minimum velocity, 1 to maximum. Velocity is converted to MIDI note-on velocity, in the range 0 to 127.
`velN(x)`	`x` = 1	Set the current note on velocity to *x* (normalized value). Velocity is converted to MIDI note-on velocity, in the range 0 to 127.
`VelrN(x)`	`x` = 1	Set the current note off (release) velocity to *x*

Music generation module commands:

		(normalized value). Velocity is converted to MIDI note-off velocity, in the range 0 to 127.
`R(x)`	x = 1	Set the current note register to x. Valid register values range from 0 to 9.
`incR(x)`	x = 1	Increment the current note register by x. The value of x may be positive or negative.
`NR(x)`	x = 1	Set the current note relative to the current register.
`key(n)`	n = 0	Set the key signature to the value specified by n.
`keyR(n)`	n = 0	Set the key signature relative to the value specified by n.
`chan(n)`	n = 1	Set the current MIDI channel to n. ($1 \le n \le 16$)
`vol(x)`	x = 1	Set MIDI volume to x (normalised). (midi controller 7)
`pan(x)`	x = 0.5	Set MIDI pan to x (normalised). 0 = left, 1 = right (midi controller 10)
`expr(x)`	x = 0	Set MIDI expression to x (normalised). (midi controller 11).

`cc(n,v)`	n = 1
	v = 0
		Set continuous controller n to v (normalised). For example `cc(10, 0.5)` is equivalent to `vol(0.5)`.

`pc(n)`	n = 1	Send MIDI program change message to current channel.
`pb(n)`	n = 0.5	Set MIDI pitch bend value to n.
`at(n)`	n = 0	Send MIDI polyphonic aftertouch message on current channel. The value of n is normalised.
`bpm(x)`	n = 120	Set the MIDI tempo to n beats per minute.
`systemx(x)`	-	Send MIDI system exclusive message.

The midi sub-system also defines a series of global identifiers corresponding to note durations. This makes incorporating timing information easier.

Symbol	Note time
WN	Whole note
HN	Half note
QN	Quarter note
EN	Eighth note
SN	Sixteenth note
TN	Thirty-second note
XN	Sixty-fourth note

References

Abelson, H. & A.A. DiSessa 1982, *Turtle Geometry: The Computer as a Medium for Exploring Mathematics,* The MIT Press Series in Artificial Intelligence, MIT Press, Cambridge, Mass.

Ackermann, P. 1993, Object-Oriented Modelling of Time Synchronisation in a Multimedia Application Framework, in *Preprints of the 95th AES Convention*, Audio Engineering Society, New York, NY.

Ackermann, P. 1996, *Developing Object-Oriented Multimedia Software,* dpunkt, Verlag für digitale Technologie, GmbH, Heidelberg, Germany.

Adam, H.-C. & K. Blossfeldt 1999, *Karl Blossfeldt, 1865-1932,* Taschen, Köln; London.

Adami, C. 1998, *Introduction to Artificial Life,* Springer, New York.

Agin, G.J. 1972, 'Representation and Description of Curved Objects', Technical Memo, No. AIM-173, October 1972. Stanford Artificial Intelligence Report, Stanford, California.

Allen, J. 1983, 'Maintaining Knowledge About Temporal Intervals', *Communications of the ACM*, **26**(11), pp. 832-843.

Angel, E. 2003, *Interactive Computer Graphics: A Top-Down Approach with OpenGL,* (3rd Edition), Addison Wesley, Boston.

Anstis, S. 1999, Illusions, in Wilson, R.A. & F.C. Keil (eds), *The MIT Encyclopaedia of the Cognitive Sciences*, The MIT Press, Cambridge, Massachusetts. pp. 385-386.

Aono, M. & T.L. Kunii 1984, 'Botanical Tree Image Generation', *IEEE Computer Graphics and Applications*, **4**(5), pp. 10-34.

Ashby, W.R. 1952, *Design for a Brain,* Chapman & Hall, London.

Ashby, W.R. 1956, *An Introduction to Cybernetics,* Chapman & Hall, London.

Ates, A.F., M. Bilgic, S. Saito & B. Sarikaya 1996, 'Using Timed CSP for Specification Verification and Simulation of Multimedia Synchronization', *IEEE Journal on Selected Areas in Communications*, **14**(1), pp. 126-137.

Audi, R. (ed.) 1999, *The Cambridge Dictionary of Philosophy*, 2nd edn, Cambridge University Press, Cambridge.

Bachelard, G. 1958, *The Poetics of Space,* (1994 Edition, with a forward by John R. Stilgoe), Beacon Press, Boston, MA.

Badler, N.I., B.A. Barsky & D. Zeltzer (eds) 1991, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, Morgan Kaufmann, San Mateo, California.

Baker, E. & M.I. Seltzer 1994, Evolving Line Drawings, in *Graphics Interface '94*, Banff, Canada. pp. 91-100.

Ball, P. 2001, *The Self-Made Tapestry: Pattern Formation in Nature,* Oxford University Press, Oxford.

Baroni, M. & C. Jacoboni 1978, *Proposal for a Grammar of Melody,* Presses de l'Université de Montréal, Montréal.

Baroni, M. & L. Callegari (eds) 1984, *Musical Grammars and Computer Analysis*, L. Olschki, Florence.

Barr, A.H., J.F. Abel, R. Barzel, D.P. Greenberg, J.C. Platt & C.W. Reynolds 1988, *Developments in Physically-Based Modeling,* Siggraph '88 Course Notes, vol. 27, ACM SIGGRAPH, Atlanta, Georgia.

Barron, F.X. 1969, *Creative Person and Creative Process,* Holt, Rinehart and Winston, Inc., New York.

Barzel, R. 1992, *Physically-Based Modeling for Computer Graphics: A Structured Approach,* Academic Press, Boston.

Baudrillard, J. 1981, *Simulacres Et Simulation,* Débats, Galilée, Paris.

Baudrillard, J. 1983, *Simulations,* Foreign Agents Series, Semiotext(e), New York.

Baudrillard, J. 2003, *Cool Memories IV 1995-2000,* Verso, London.

Beckermann, A. 1992, Reductive and Nonreductive Physicalism, in Beckermann, A., H. Flohr & J. Kim (eds), *Emergence or Reduction?: Essays on the Prospects of Nonreductive Physicalism*, W. de Gruyter, Berlin; New York. pp. 1-21.

Beckermann, A., H. Flohr & J. Kim 1992, *Emergence or Reduction?: Essays on the Prospects of Nonreductive Physicalism,* Foundations of Communication and Cognition (Grundlagen Der Kommunikation Und Kognition), W. de Gruyter, Berlin; New York.

Bell, B. 1992, Music Structures: Interleaving the Temporal and Hierarchical Aspects in Music, in Balaban, M., K. Ebcioglu & O. Laske (eds), *Understanding Music with AI*, MIT Press and AAAI Press, Cambridge, MA and Menlo Park, CA. pp. 110-139.

Bell, B. & J. Kippen 1992, Bol Processor Grammars, in Balaban, M., K. Ebcioglu & O. Laske (eds), *Understanding Music with AI*, MIT Press and AAAI Press, Cambridge, MA and Menlo Park, CA. pp. 336-401.

Bell, Q. 1976, *On Human Finery,* (Second Edition), Hogarth Press, London.

Benjamin, W. & H. Arendt 1968, *Illuminations,* (1st Edition), Harcourt Brace & World, New York.

Bentley, P.J. & D.W. Corne (eds) 2002, *Creative Evolutionary Systems*, Academic Press, London.

Berger, J. 1972, *Ways of Seeing: Based on the BBC Television Series with John Berger,* British Broadcasting Corporation; Penguin, London; Harmondsworth.

Berger, J. 1980, *About Looking,* Vintage International, New York, NY.

Berlekamp, E.R., J.H. Conway & R.K. Guy 1982, *Winning Ways for Your Mathematical Plays,* vol. 2, Academic Press, New York.

Bertalanffy, L.v. 1968, *General System Theory : Foundations, Development, Applications,* (Revised Edition), International Library of Systems Theory and Philosophy, G. Braziller, New York.

Bertino, E. & E. Ferrari 1998, 'Temporal Synchronization Models for Multimedia Data', *IEEE TKDE*, **10**(4), pp. 612-631.

Beyls, P. 1989, The Musical Universe of Cellular Automata, in Wells, T. & D. Butler (eds), *Proceedings of the 1989 International Computer Music Conference*, International Computer Music Association, San Francisco. pp. 34-41.

Birkhoff, G.D. 1933, *Aesthetic Measure,* Harvard University Press, Cambridge, MA.

Blakowski, G. & R. Steinmetz 1996, 'A Media Synchronization Survey: Reference Model, Specification and Case Studies', *IEEE Journal on Selected Areas in Communications*, **14**(1), pp. 5-35.

Blinn, J.F. 1982, 'A Generalization of Algebraic Surface Drawing', *ACM Transactions on Graphics*, **1**(3), pp. 235-256.

Blitz, D. 1992, *Emergent Evolution: Qualitative Novelty and the Levels of Reality,* Kluwer Academic Publishers, Dordrecht.

Bloomenthal, J. 1985, Modeling the Mighty Maple. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22-26, 1985, Barsky, B.A., ed). In *Computer Graphics* **19**(3) ACM SIGGRAPH, New York, pp. 305-311.

Bloomenthal, J. 1990, Calculation of Reference Frames Along a Space Curve, in Glassner, A. (ed) *Graphics Gems*, Academic Press, Boston. pp. 567-571.

Bloomenthal, J. 1995, *Skeletal Design of Natural Forms,* Ph.D. thesis, Department of Computer Science, University of Calgary, Calgary, Alberta.

Bloomenthal, J. & K. Shoemake 1991, Convolution Surfaces. Proceedings of SIGGRAPH '91 (Las Vegas, Nevada, July 28 - August 2, 1991). In *SIGGRAPH '91* **25**(4) ACM SIGGRAPH, New York, pp. 251-256.

Blossfeldt, K. & R. Sachsse 1994, *Karl Blossfeldt: Photographs 1865-1932,* Benedikt Taschen, Köln.

Boden, M.A. 1994, What Is Creativity?, in Boden, M.A. (ed) *Dimensions of Creativity*, MIT Press, Cambridge, MA. pp. 75-117.

Boden, M.A. 1996, *The Philosophy of Artificial Life,* Oxford Readings in Philosophy, Oxford University Press, Oxford; New York.

Böhm, W., G. Farin & J. Kahmann 1984, 'A Survey of Curve and Surface Methods in CAGD', *Computer Aided Geometric Design*, **1**(1), pp. 1-60.

Bolter, J.D. & R. Grusin 1999, *Remediation: Understanding New Media,* MIT Press, Cambridge, Mass.

Bonabeau, E.W. & G. Theraulaz 1994, 'Why Do We Need Artificial Life?' *Artificial Life*, **1**(3), pp. 303-325.

Bowcott, P. 1989, Cellular Automata as a Means of High Level Compositional Control of Granular Synthesis, in Wells, T. & D. Butler (eds), *Proceedings of the 1989 International Computer Music Conference*, San Francisco. pp. 55-57.

Breidbach, O. 1998, Brief Instructions to Viewing Haeckel's Pictures, in Ashdown, M. (ed) *Art Forms in Nature: The Prints of Ernst Haeckel*, Prestel-Verlag, Munich. pp. 9-18.

Bringhurst, R. 1992, *The Elements of Typographic Style,* (Second Edition), Hartley & Marks, Vancouver, BC.

Brown, D.E. 1991, *Human Universals,* McGraw-Hill, New York.

Brown, R. 2001, *Biotica: Art, Emergence and Artificial Life,* RCA CRD Research Publications, Royal College of Art, London.

Brumbaugh, R.S. 1981, *The Philosophers of Greece,* State University of New York Press, Albany, N.Y.

Buchanan, M.C. & P.T. Zellweger 1992, 'Specifying Temporal Behaviour in Hypermedia Documents', *European Conference on Hypertext '92 (Proceedings of the ACM Conference on Hypertext)*, Milan, Italy, pp. 262-271.

Buchanan, M.C. & P.T. Zellweger 1993, Automatic Temporal Layout Mechanisms, in *Computer Graphics (ACM Multimedia '93 Proceedings)*, Addison-Wesley pp. 341-350.

Bulhak, A. 1999, Evolving Automata for Dynamic Rearrangement of Sampled Rhythm Loops, in Dorin, A. & J. McCormack (eds), *First Iteration: A Conference on Generative Systems in the Electronic Arts*, CEMA, Melbourne, Australia. pp. 46-54.

Burnham, J. 1968a, *Beyond Modern Sculpture: The Effects of Science and Technology on the Sculpture of This Century,* G. Braziller; Allen Lane, The Penguin Press, New York, London.

Burnham, J. 1968b, 'Systems Esthetics', *Artforum*, **7**(1), pp. 30-35.

Burnham, J. 1986, Art and Technology: The Panacea That Failed, in Hanhardt, J.G. (ed) *Video Culture: A Critical Investigation*, G.M. Smith Peregrine Smith Books in association with Visual Studies Workshop Press, Layton, Utah. pp. 232-248.

Burnstein, L. 1976, *The Unanswered Question,* Harvard University Press, Cambridge, MA.

Buxton, W., W. Reeves, R. Baeker & L. Mezei 1978, 'The Use of Hierarchy and Instance in a Data Structure for Computer Music', *Computer Music Journal*, **2**(2), pp. 10-20.

Buxton, W., W. Reeves, R. Baeker & L. Mezei 1985, The Use of Hierarchy and Instance in a Data Structure for Computer Music, in Roads, C. & J. Strawn (eds), *Foundations of Computer Music*, MIT Press, Cambridge, MA. pp. 443-466.

Cahoone, L.E. 1996, *From Modernism to Postmodernism: An Anthology,* Blackwell Publishers, Cambridge, Mass.

Calvert, T. 1991, Composition of Realistic Animation Sequences for Multiple Human Figures, in Badler, N.I., B.A. Barsky & D. Zeltzer (eds), *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, Morgan Kaufmann, San Mateo, CA. pp. 35-50.

Cardew, C. 1971, *The Great Learning: The First Chapter of the Confucian Classic with Music in 7 Paragraphs*, Experimental Music Catalogue, London.

Cardew, C. (ed.) 1972, *Scratch Music*, Latimer New Dimensions Ltd., London.

Cariani, P. 1991, Emergence and Artificial Life, in Langton, C.G., C. Taylor, D. Farmer & S. Rasmussen (eds), *Artificial Life II, SFI Studies in the Sciences of Complexity*, Vol. 10, Addison-Wesley, Redwood City, CA. pp. 775-797.

Carpenter, L.C. 1980, Computer Rendering of Fractal Curves and Surfaces. (Seattle, Washington). In *7th Annual conference on Computer Graphics and Interactive Techniques (SIGGRAPH '80)* ACM SIGGRAPH, Seattle, Washington, p. 109.

Chaitin, G.J. 1999, *The Unknowable,* Springer Series in Discrete Mathematics and Theoretical Computer Science, Springer, Singapore; New York.

Chaitin, G.J. 2002, *Conversations with a Mathematician: Math, Art, Science, and the Limits of Reason: A Collection of His Most Wide-Ranging and Non-Technical Lectures and Interviews,* Springer, London; New York.

Chareyron, J. 1990, 'Digital Synthesis of Self-Modifying Waveforms by Means of Linear Automata', *Computer Music Journal*, **14**(4), pp. 25-41.

Chase, L. & R. Goings 1988, *Ralph Goings: Essay/Interview,* Abrams, New York.

Chemillier, M. 1992, Automata and Music, in Strange, A. (ed) *Proceedings of the 1992 International Computer Music Conference*, International Computer Music Association, San Francisco. pp. 370-371.

Chen, X. & P. Lienhardt 1992, 'Modelling and Programming Evolutions of Surfaces', *Computer Graphics Forum*, **11**(5), pp. 323-341.

Chomsky, N. 1956, 'Three Models for the Description of Language', *IRE Transactions on Information Theory*, **2**(3), pp. 113-124.

Chomsky, N. 1957, *Syntactic Structures,* Mouton, The Hague.

Chomsky, N. 1963, Formal Properties of Grammars, in Luce, R.D., R.R. Bush & E. Galanter (eds), *Handbook of Mathematical Psychology*, Vol. 2, Wiley, New York. pp. 323-418.

CipherDigital 1987, *The Time Code Handbook,* (Second Edition), Cipher Digital Inc., Frederick, MD, USA.

Clark, T. 2002, *Martin Heidegger,* Routledge Critical Thinkers, Routledge, New York.

Coates, P., T. Broughton & H. Jackson 1999, Exploring Three-Dimensional Design Worlds Using Lindenmayer Systems and Genetic Programming, in Bentley, P.J. (ed) *Evolutionary Design by Computers*, Morgan Kaufmann, London, UK. p. Chapter 14.

Collings, M. 2000, *This Is Modern Art,* Seven Dials, London.

Cook, N. 1987, *A Guide to Musical Analysis,* J. M. Dent, London.

Cook, R.L. 1986, 'Stochastic Sampling in Computer Graphics', *ACM Transactions on Graphics*, **5**(1), pp. 51-72.

Cook, R.L. & K.E. Torrance 1992, 'A Reflection Model for Computer Graphics', *ACM Transactions on Graphics*, **1**(1), pp. 7-24.

Cook, R.L., L. Carpenter & E. Catmull 1987, The Reyes Image Rendering Architecture. Proceedings of SIGGRAPH '87 (Anaheim, California, July 27-31, 1987, Stone, M.C., ed). In *Computer Graphics* **21**(4) ACM SIGGRAPH, New York, pp. 95-102.

Cope, D. 1991, *Computers and Musical Style,* The Computer Music and Digital Audio Series, vol. 6, Oxford University Press, Oxford.

Copland, A. 1988, *What to Listen for in Music,* Mentor, New York.

Cordle, D. 1999, *Postmodern Postures: Literature, Science and the Two Cultures Debate,* Ashgate, Aldershot.

Coyne, R. 1999, *Technoromanticism: Digital Narrative, Holism, and the Romance of the Real,* MIT Press, Cambridge, Mass.; London.

Crutchfield, J.P. 1994, 'The Calculi of Emergence: Computation, Dynamics, and Induction', *Physica D*, (special issue on the Proceedings of the Oji International Seminar Complex Systems – from Complex Dynamics to Artificial Reality).

Crutchfield, R.S. 1973, The Creative Process, in Bloomberg, M. (ed) *Creativity: Theory and Research*, College and University Press pp. 54-74.

Curry, R. 1999, 'On the Evolution of Parametric L-Systems', Technical Report, No. 1999-644-07, 9 November 1999. Department of Computer Science, University of Calgary, Calgary.

Dartnall, T. (ed.) 2002, *Creativity, Cognition, and Knowledge: An Interaction*, Perspectives on Cognitive Science, Praeger, Westport, Connecticut; London.

Darwin, C.R. 1859, *On the Origin of Species,* (Reprinted 1968, Edited by J.W. Burrow, Penguin Classics), John Murray, London.

Dawkins, R. 1982, *The Extended Phenotype: The Gene as the Unit of Selection,* Freeman, Oxford [Oxfordshire]; San Francisco.

Dawkins, R. 1986, *The Blind Watchmaker,* Longman Scientific & Technical, Essex, UK.

De Leon, M.K. 1990a, 'Visualization of Environmental Effects on Branching Objects', *Resource Technology '90*, Washington D.C., pp. 90-99.

De Leon, M.K. 1990b, *I Have Never Seen, but I Know...* in ACM SIGGRAPH Video Review Issue 66 SIGGRAPH '90 Film & Video Theater, ACM SIGGRAPH, New York.

De Leon, M.K. 1991, 'Branching Object Generation and Animation System with Cubic Hermite Interpolation', *The Journal of Visualization and Computer Animation*, **2**(2), pp. 60-67.

De Leon, M.K. 2002, *Midori Kitagawa's Gallery* (Web page), <http://www.accad.ohio-state.edu/~midori/gallery.html> (Accessed 31 May 2002).

de Reffye, P., C. Edelin, J. Françon, M. Jaeger & C. Puech 1988, Plant Models Faithful to Botanical Structure and Development. Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988). In *Computer Graphics* **22**(4) ACM SIGGRAPH, New York, pp. 151-158.

Denavit, J. & R. Hartenberg 1955, 'A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices', *Journal of Applied Mechanics*, **77**(June), pp. 215-221.

Dennett, D.C. 1984, *Elbow Room: The Varieties of Free Will Worth Wanting,* MIT Press, Cambridge, Mass.

Dennett, D.C. 1991, 'Real Patterns', *Journal of Philosophy*, **88**, pp. 27-51.

Deussen, O., P. Hanrahan, B. Lintermann, R. Mech, M. Pharr & P. Prusinkiewicz 1998, Realistic Modeling and Rendering of Plant Ecosystems. Proceedings of SIGGRAPH 98 (Orlando, Florida, July 19-24, 1998). In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 275-286.

Devroye, L. 1986, *Non-Uniform Random Variate Generation,* Springer-Verlag, New York.

Diamond, J.M. 1992, *The Third Chimpanzee: The Evolution and Future of the Human Animal,* (1st Edition), HarperCollins, New York, NY.

Dissanayake, E. 1988, *What Is Art For?,* University of Washington Press, Seattle.

Dissanayake, E. 1995, *Homo Aestheticus: Where Art Comes from and Why,* University of Washington Press, Seattle.

Doczi, G. 1981, *The Power of Limits: Proportional Harmonies in Nature, Art and Architecture,* Shambhala (Distributed by Routledge & Kegan Paul), London.

Döllner, J. & K. Hinrichs 1997, 'Object-Oriented 3D Modelling, Animation and Interaction', *Journal of Visualization and Computer Animation*, **8**, pp. 33-64.

Dolson, M. 1989, 'Machine Tongues XII: Neural Networks', *Computer Music Journal*, **13**(4), pp. 28-40.

Dorin, A. 2001a, 'Aesthetic Fitness and Artificial Evolution for the Selection of Imagery from the Mythical Infinite Library' in Kelemen, J. & P. Sosík (eds), *Advances in Artificial Life, Proceedings of the 6th European Conference on Artificial Life*, vol. LNAI2159, Springer-Verlag, Prague, pp. 659-668.

Dorin, A. (ed.) 2001b, *Second Iteration: Conference on Generative Systems in the Electronic Arts*, Centre for Electronic Media Art (CEMA), Monash University, Melbourne.

Dorin, A. & J. McCormack (eds) 1999, *First Iteration: A Conference on Generative Systems in the Electronic Arts*, Centre for Electronic Media Art (CEMA), Monash University, Melbourne.

Dorin, A. & J. McCormack 2001, 'First Iteration / Generative Systems (Guest Editor's Introduction)', *Leonardo*, **34**(3), p. 335.

Douady, S. & Y. Couder 1992, 'Phyllotaxis as a Physical Self-Organized Growth Process', *Phys. Rev. Lett.*, **68**, pp. 2098-2101.

Douady, S. & Y. Couder 1996, 'Phyllotaxis as a Dynamical Self Organizing Process (Part I, II, III)', *Journal of Theoretical Biology*, **139**, pp. 178-312.

Dreyfus, H.L. 1991, *Being-in-the-World: A Commentary on Heidegger's Being and Time, Division I,* MIT Press, Cambridge, Mass.

Driessens, E. & M. Verstappen 2001a*, Ima Traveller* (website), <http://www.xs4all.nl/%7Enotnot/ima/IMAtraveller.html> (Accessed 7 October 2001).

Driessens, E. & M. Verstappen 2001b*, Not Not (Artist's Home Page)* (website), <http://www.xs4all.nl/%7Enotnot/> (Accessed 7 October 2001).

Duhem, P. 1906, *The Aim and Structure of Physical Theory,* (1954 edition), Princeton University Press, Princeton.

Duprat, H. & C. Besson 1998, 'The Wonderful Caddis Worm: Sculptural Work in Collaboration with Trichoptera', *Leonardo*, **31**(3), pp. 173-182.

Durikovic, R., K. Kaneda & H. Yamashita 1998, 'Animation of Biological Organ Growth Based on L-Systems', *Computer Graphics Forum (EUROGRAPHICS '98)*, **17**(3), pp. 1-14.

Ebert, D.S., F.K. Musgrave, D. Peachey & K. Perlin 1993, *Procedural Modeling and Rendering Techniques,* Siggraph 93 Course Notes, vol. 44, ACM SIGGRAPH, Anaheim, California.

Ebert, D.S., F.K. Musgrave, D. Peachey, K. Perlin & S. Worley 1994, *Texturing and Modeling: A Procedural Approach,* Academic Press, London.

Ebert, D.S., F.K. Musgrave, D. Peachey, K. Perlin & S. Worley 2003, *Texturing & Modeling: A Procedural Approach,* (Third Edition), Morgan Kaufmann, San Francisco, CA.

Ede, S. (ed.) 2000, *Strange and Charmed: Science and the Contemporary Visual Arts*, Calouste Gulbenkian Foundation, London.

Eichhorst, P. & W.J. Savitch 1980, 'Growth Functions of Stochastic Lindenmayer Systems', *Information and Control*, **45**, pp. 217-228.

Elam, K. 2001, *Geometry of Design: Studies in Proportion and Composition,* Design Briefs, Princeton Architectural Press, New York, N.Y.

Ellis, W.D. 1938, *A Source Book of Gestalt Psychology,* Routledge & K. Paul, London.

Emmeche, C. 1994, *The Garden in the Machine,* Princeton University Press, Princeton, NJ.

Emmeche, C., S. Køppe & F. Stjernfelt 1997, 'Explaining Emergence: Towards an Ontology of Levels', *Journal for General Philosophy of Science*, **28**, pp. 83-119.

Erickson, R.O. 1983, The Geometry of Phyllotaxis, in Dale, J.E. & F.L. Milthorpe (eds), *The Growth and Functioning of Leaves*, Cambridge University Press, Cambridge. pp. 53-88.

Faith, J. 2000, *Emergent Representations: Dialectical Materalism and the Philosophy of Mind,* D.Phil thesis, COGS, University of Sussex, Sussex.

Farin, G. 1990, *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide,* (Second Edition), Academic Press, London.

Faux, I.D. & M.J. Pratt 1979, *Computational Geometry for Design and Manufacture,* Ellis Horwood, Chichester.

Featherstone, R. 1988, *Robot Dynamics Algorithms,* Kluwer Academic Publishers, Norwell, MA.

Fenton, T. 1969, 'Two Contributions to the Art and Science Muddle: 1. Constructivism and Its Confusions', *Artforum*, **7**(5), pp. 22-27.

Feyerabend, P.K. 1975, *Against Method: Outline of an Anarchistic Theory of Knowledge,* Nlb; Humanities Press, London; Atlantic Highlands.

Feyerabend, P.K. 1985, Attempt at a Realistic Interpretation of Experience, in Feyerabend, P.K. (ed) *Philosophical Papers*, Vol. 1, Cambridge University Press, Cambridge. pp. 17-36.

Fine, A. 1986a, *The Shaky Game: Einstein, Realism, and the Quantum Theory,* Science and Its Conceptual Foundations, University of Chicago Press, Chicago.

Fine, A. 1986b, 'Unnatural Attitudes: Realist and Instrumentalist Attachments to Science', *Mind*, **95**, pp. 149-179.

Fine, A. 1999, Realism and Antirealism, in Wilson, R.A. & F.C. Keil (eds), *The MIT Encyclopaedia of the Cognitive Sciences*, MIT Press, Boston, Massachusetts. pp. 707-709.

Flake, G.W. 1998, *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation,* MIT Press, Cambridge, Mass.

Fleischer, K.W. & A.H. Barr 1994, A Simulation Testbed for the Study of Multicellular Development: The Multiple Mechanisms of Morphogenesis, in Langton, C.G. (ed) *Artificial Life III*, Vol. XVII, Addison-Wesley, Reading, Massachusetts. pp. 389-416.

Fleischer, K.W., D.H. Laidlaw, B.L. Currin & A.H. Barr 1995, Cellular Texture Generation. Proceedings of SIGGRAPH 95 (Los Angeles, California, August 6-11, 1995). In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 239-248.

Foley, J.D., A.v. Dam, S.K. Feiner & J.F. Hughes 1990, *Computer Graphics: Principles and Practice,* (Second Edition), Addison-Wesley, Reading MA.

Foley, J.D., A. van Dam, S.K. Feiner, J.F. Hughes & R.L. Phillips 1994, *Introduction to Computer Graphics,* Addison-Wesley, Reading, Mass.

Forrest, S. 1990, Emergent Computation: Self-Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks, in Forrest, S. (ed) *Emergent Computation*, A Special Issue of Physica D, MIT Press, Cambridge, Mass. pp. 1-11.

Foster, F. 1999, *The Story of Computer Graphics* in ACM SIGGRAPH Video Review Series, ACM SIGGRAPH, New York.

Foster, H. 1996, *The Return of the Real: The Avant-Garde at the End of the Century,* MIT Press, Cambridge, Mass.

Fournier, A. 1987, Prolegomenon, in *The Modeling of Natural Phenomena*, Siggraph '87 Course Notes, Vol. 16, ACM SIGGRAPH, Anaheim, California. pp. 3-37.

Fournier, A. & W.T. Reeves 1986, A Simple Model of Ocean Waves. Proceedings of SIGGRAPH '86 (Dallas, Texas, August 18-22, 1986, Evans, D.C. & R.J. Athay, eds). In *Computer Graphics* **20**(4) ACM SIGGRAPH, New York, pp. 75-84.

Fournier, A., D. Fussell & L. Carpenter 1982, 'Computer Rendering of Stochastic Models', *Communications of the ACM*, **25**(6), pp. 371-384.

Fournier, A., J. Bloomenthal, P. Oppenheimer, W.T. Reeves & A.R. Smith 1987, *The Modelling of Natural Phenomena,* Siggraph '87 Course Notes, vol. 16, ACM SIGGRAPH, Anaheim, CA.

Fowler, D.R., P. Prusinkiewicz & J. Battjes 1992, A Collision-Based Model of Spiral Phyllotaxis. Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26-31, 1992). In *Computer Graphics* **26**(2) ACM SIGGRAPH, New York, pp. 361-368.

Fowler, D.R., H. Meinhardt & P. Prusinkiewicz 1992, Modeling Seashells. Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26-31, 1992). In *Computer Graphics* **26**(2) ACM SIGGRAPH, New York, pp. 379-387.

Francblin, C. & J. Baudrillard 1991, *Art and Philosophy: Baudrillard, Gadamer, Jameson, Kristeva, Lyotard, Marin, Perniola, Sloterdijk, Sollers, Virilio, West,* New Criticism Series; No. 2, G. Politi, Milan.

Freeland, C.A. 2001, *But Is It Art?: An Introduction to Art Theory,* Oxford University Press, Oxford.

Frijters, D. & A. Lindenmayer 1974, A Model for the Growth and Flowering of *Aster Novae-Angliae* on the Basis of Table (1,0)L-Systems, in Rozenberg, G. & A. Salomaa (eds), *L Systems*, Lecture Notes in Computer Science, Vol. 15, Springer-Verlag, Berlin. pp. 24-52.

Gamma, E. 1995, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley Professional Computing Series, Addison-Wesley, Reading, Mass.

Gardner, G.Y. 1984, Simulation of Natural Scenes Using Textured Quadric Surfaces. Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July 23-27, 1984, Christiansen, H., ed). In *Computer Graphics* **18**(3) ACM SIGGRAPH, New York, pp. 11-20.

Gardner, G.Y. 1985, Visual Simulation of Clouds. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22-26, 1985, Barsky, B.A., ed). In *Computer Graphics* **19**(3) ACM SIGGRAPH, New York, pp. 297-303.

Gardner, G.Y. 1990, 'Forest Fire Simulation', *Computer Graphics*, **24**(4), p. 430.

Gervautz, M. & C. Traxler 1994, 'Representation and Realistic Rendering of Natural Phenomena with Cyclic CSG-Graphs', Technical Report, No. TR-186-2-94-19, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Wien, Austria.

Giere, R.N. 1999, Using Models to Represent Reality, in Magnani, L., N.J. Nersessian & P. Thagard (eds), *Model-Based Reasoning in Scientific Discovery*, Kluwer Academic/Plenum Publishers, New York. pp. 41-57.

Girard, M. 1991, Constrained Optimization of Articulated Animal Movement, in Badler, N.I., B.A. Barsky & D. Zeltzer (eds), *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, Morgan Kaufmann, San Mateo, CA. pp. 209-232.

Girard, M. & A.A. Maciejewski 1985, Computational Modeling for the Computer Animation of Legged Figures. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22-26, Barsky, B.A., ed). In *Computer Graphics* **19**(3) ACM SIGGRAPH, New York, pp. 263-270.

Gjerdingen, R. 1989, 'Using Connectionist Models to Explore Complex Musical Patterns', *Computer Music Journal*, **13**(3), pp. 67-75.

Globus, G.G. 1995, *The Postmodern Brain,* Advances in Consciousness Research, vol. 1, J. Benjamins Pub. Co., Amsterdam; Philadelphia.

Goldberg, D.E. 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley, Reading, MA.

Gombrich, E.H. 1984, *The Story of Art,* (14th Edition), Phaidon, Oxford.

Goodman, N. 1968, *Languages of Art; an Approach to a Theory of Symbols,* Bobbs-Merrill, Indianapolis.

Graf, J. & W. Banzhaf 1995, Interactive Evolution of Images, in McDonnell, J.R., R.G. Reynolds & D.B. Fogel (eds), *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pp. 53-65.

Gray, J. 2002, *Straw Dogs: Thoughts on Humans and Other Animals,* Granta Books, London.

Greig, D. 1999, *Field Guide to Australian Wildflowers,* New Holland Publishers (Australia) Pty. Ltd., Sydney.

Grenfenstette, J.J. (ed.) 1985, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Grenfenstette, J.J. (ed.) 1987, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Grimmet, G.R. & D.R. Stirzaker 1982, *Probability and Random Process,* Oxford University Press, Oxford.

Haeckel, E. 1998, *Art Forms in Nature: The Prints of Ernst Haeckel,* Prestel-Verlag, Munich.

Hagen, M.A. 1986, *Varieties of Realism: Geometries of Representational Art,* Cambridge University Press, Cambridge ; New York.

Hanan, J. 1992, *Parametric L-Systems and Their Application to the Modelling and Visualization of Plants,* Ph.D. thesis, Computer Science, University of Regina, Saskatchewan.

Hanrahan, P. 1983, Ray Tracing Algebraic Surfaces. Proceedings of SIGGRAPH '83 (Detroit, Michigan, July 25-29, 1983, Christiansen, H., ed). In *Computer Graphics* **17**(3) ACM SIGGRAPH, New York, pp. 83-90.

Hansen, N.R. 1958, *Patterns of Discovery,* Cambridge University Press.

Hart, J.C. 1996, 'On Efficiently Representing Procedural Geometry'.

Hayles, N.K. 1996, Narratives of Artificial Life, in Robertson, G., M. Mash, L. Tickner, J. Bird, B. Curtis & T. Putnam (eds), *Futurenatural: Nature, Science, Culture*, Routledge, New York and London. pp. 146-164.

Hayles, N.K. 2001, 'Book Review: Silicon Second Nature: Culturing Artificial Life in a Digital World.' *Artificial Life*, **7**(4), pp. 425-428.

Heckbert, P.S. 1987, Ray Tracing Jell-O Brand Gelatin. Proceedings of SIGGRAPH '87 (Anaheim, California, 27-31 July, 1997, Stone, M.C., ed). In *Computer Graphics* **21**(4) ACM SIGGRAPH, New York, pp. 73-74.

Helmreich, S. 2000, *Silicon Second Nature: Culturing Life in a Digital World,* (Updated edition (June 2000)), University of California Press.

Hempel, C.G. & P. Oppenheim 1948, 'Studies in the Logic of Explanation', *Philosophy of Science*, **15**, pp. 135-175.

Herman, G.T. & G. Rozenberg 1975, *Developmental Systems and Languages,* North-Holland, Amsterdam.

Hewlett, W.B. & E. Selfridge-Field 1997, MIDI, in Selfridge-Field, E. (ed) *Beyond MIDI: The Handbook of Musical Codes*, MIT Press, Cambridge, Massachusetts. pp. 41-72.

Hiller, L. 1981, 'Composing with Computers: A Progress Report', *Computer Music Journal*, **5**(4), pp. 7-21.

Hiller, L.A. & L. Issacson 1959, *Experimental Music,* McGraw-Hill, New York.

Hogeweg, P. & B. Hesper 1974, 'A Model Study on Biomorphological Description', *Pattern Recognition*, **6**, pp. 165-179.

Holland, J.H. 1992, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence,* (1st MIT Press Edition), Complex Adaptive Systems, MIT Press, Cambridge, Mass.

Holland, J.H. 1995, *Hidden Order: How Adaptation Builds Complexity,* Helix Books, Addison-Wesley, Reading, Mass.

Holton, M. 1994, 'Strands, Gravity and Botanical Tree Imagery', *Computer Graphics Forum*, **13**(1), pp. 57-67.

Holtzman, S.R. 1980, 'A Generative Grammar Definition Language for Music', *Interface*, **9**(2), pp. 1-48.

Holtzman, S.R. 1981, 'Using Generative Grammars for Music Composition', *Computer Music Journal*, **5**(1), pp. 51-64.

Horkheimer, M. 1937, Traditional and Critical Theory, in *Critical Theory: Selected Essays*, Seabury Press, New York, NY. pp. 188-243.

Horn, M.K. 1983, 'Fourth and Fifth-Order Scaled Runge-Kutta Algorithms for Treating Dense Output', *SIAM Journal of Numerical Analysis*, **20**, pp. 558-568.

Hornby, G.S. & J.B. Pollack 2001a, 'Evolving L-Systems to Generate Virtual Creatures', *Computers & Graphics*, **26**(6), pp. 1041-1048.

Hornby, G.S. & J.B. Pollack 2001b, The Advantages of Generative Grammatical Encodings for Physical Design, in *Proceedings of the 2001 Congress on Evolutionary Computation*, IEEE Press pp. 600-607.

Horner, A. & D.E. Goldberg 1991, 'Genetic Algorithms and Computer-Assisted Music Composition', Technical Report, No. CCSR-91-20, University of Illinois, Chicago.

House, D.H., G.S. Schmidt, S.A. Arvin & M.K. De Leon 1998, 'Visualizing a Real Forrest', *IEEE Computer Graphics & Applications*, **18**(1), pp. 12-15.

Humphrey, N.K. 1973, 'The Illusion of Beauty', *Perception*, **2**, pp. 429-439.

Inakage, M., D. Peachey, G.Y. Gardner, A. Fournier & K. Perlin 1988, *Functional Based Modeling,* Siggraph '88 Course Notes, vol. 28, ACM SIGGRAPH, Atlanta, Georgia.

Jackson, H. 2002, Toward a Symbiotic Coevolutionary Approach to Architecture, in Bentley, P.J. & D.W. Corne (eds), *Creative Evolutionary Systems*, Academic Press, London. pp. 299-313.

Jacob, C. 1994, Genetic L-System Programming, in Davidor, Y., H.-P. Schwefel & R. Männer (eds), *Parallel Problem Solving from Nature III*, LNCS, Vol. 866, Springer-Verlag, Berlin. pp. 334-343.

Jacob, C. 1995, 'Genetic L-System Programming: Breeding and Evolving Artificial Flowers with Mathematica', *IMS '95 First International Mathematica Symposium*, Computational Mechanics Publications, Southampton, UK, pp. 215-222.

Jacob, C. 1996, Evolving Evolution Programs: Genetic Programming and L-Systems, in Koza, J.R., D.E. Goldberg, D.B. Fogel & R.L. Riolo (eds), *Genetic Programming 1996: Proceedings of the First Annual Conference*, MIT Press, Cambridge, MA. pp. 28-31.

Jankel, A. & R. Morton 1984, *Creative Computer Graphics,* Cambridge University Press, Cambridge.

Jay, M. 1993, *Downcast Eyes: The Denigration of Vision in Twentieth-Century French Thought,* University of California Press, Berkeley.

Jean, R.V. 1982, *Mathematical Approach to Pattern & Form in Plant Growth,* Wiley, New York.

Jean, R.V. & D. Barabe 1998, *Symmetry in Plants,* World Scientific Series in Mathematical Biology and Medicine, vol. 4, World Scientific.

Johnson-Laird, P.N. 1993, *Human and Machine Thinking,* Lawrence Eribaum Associates.

Jones, K. 1981, 'Compositional Applications of Stochastic Processes', *Computer Music Journal*, **5**(2), pp. 381-397.

Kajiya, J.T. 1983, New Techniques for Ray Tracing Procedurally Defined Surfaces. Proceedings of SIGGRAPH '83 (Detroit, Michigan, July 25-29, 1983, Christiansen, H., ed). In *Computer Graphics* **17**(3) ACM SIGGRAPH, New York, pp. 91-102.

Kajiya, J.T. & B.P. Von Herzen 1984, Ray Tracing Volume Densities. Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July 23-27, 1984, Christiansen, H., ed). In *Computer Graphics* **18**(3) ACM SIGGRAPH, New York, pp. 165-175.

Karolyi, O. 1965, *Introducing Music,* (1991 Reprinted Edition), Penguin, London, England.

Kawaguchi, Y. 1982, A Morphological Study of the Form of Nature. Proceedings of the Ninth Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '82) (Boston, Massachusetts, July 26-30, 1982, Bergeron, R.D., ed). In *Computer Graphics* **16**(3) ACM SIGGRAPH, New York, pp. 223-232.

Kawaguchi, Y. 1985, *Growth Morphogenesis — a Journey to the Origins of Form,* JICC Publishing Inc., Tokyo, Japan.

Kay, A.C. 1993, 'The Early History of Smalltalk', *The second ACM SIGPLAN conference on History of programming languages*, ACM Press, Cambridge, Massachusetts, pp. 69-95.

Kellman, P.J. & E.S. Spelke 1983, 'Perception of Partly Occluded Objects in Infancy', *Cognitive Psychology*, **15**, pp. 483-524.

Kent, J.R., W.E. Carson & R.E. Parent 1992, Shape Transformation for Polyhedral Objects. Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26-31, 1992). In *Computer Graphics* **26**(2) ACM SIGGRAPH, New York, pp. 47-54.

Kepes, G. 1944, *Language of Vision,* P. Theobald, Chicago, IL.

Kernighan, B.W. & D.M. Ritchie 1988, *The C Programming Language,* (Second Edition), Prentice Hall, Englewood Cliffs, New Jersey.

Kincaid, H. 1988, 'Supervenience and Explanation', *Synthese*, **77**, pp. 251-281.

Kitano, H. 1990, 'Designing Neural Networks Using Genetic Algorithms with Graph Generation System', *Complex Systems*, **4**(4), pp. 461-476.

Klok, F. 1986, 'Two Moving Coordinate Frames for Sweeping Along a 3D Trajectory', *Computer Aided Geometric Design*, **3**(3), pp. 217-229.

Koffka, K. 1935, *Principles of Gestalt Psychology,* Harcourt Brace, New York.

Kókai, G., Z. Tóth & R. Ványi 1999, Evolving Artificial Trees Described by Parametric L-Systems, in *Proceedings of the First Canadian Workshop on Soft Computing*, Edmonton, Alberta, Canada. pp. 1722-1728.

Korein, J.U. & N.I. Badler 1982, 'Techniques for Generating the Goal-Directed Motion of Articulated Structures', *IEEE Computer Graphics & Applications*, **2**(9), pp. 71-81.

Koza, J.R. 1990, 'Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems', Technical Report, No. STAN-CS-90-1314, June 1990. Stanford University Computer Science Department.

Koza, J.R. 1992, *Genetic Programming: On the Programming of Computers by Means of Natural Selection,* Complex Adaptive Systems, MIT Press, Cambridge, Mass.

Krausse, J. & C. Lichtenstein (eds) 1999, *Your Private Sky: R. Buckminster Fuller, the Art of Design Science*, Lars Müller Publishers, Baden, Switzerland.

Kreyszig, E. 1999, *Advanced Engineering Mathematics,* (8th Edition), John Wiley & Sons, New York.

Kuhn, T.S. 1996, *The Structure of Scientific Revolutions,* (Third Edition), University of Chicago Press, Chicago, Ill.

Langer, S.K. 1948, *Philosophy in a New Key: A Study in the Symbolism of Reason, Rite, and Art,* New American Library, New York.

Langston, P.S. 1986, '201 644-2332 or Eedie & Eddie on the Wire: An Experiment in Music Generation', *USENIX Summer Conference*, USENIX Association, Altanta, GA, USA, pp. 13-27.

Langston, P.S. 1990, 'Little Languages for Music', *Computing Systems*, **3**(2), pp. 193-288.

Langton, C.G. 1989, Artificial Life, in Langton, C.G. (ed) *Artificial Life, SFI Studies in the Sciences of Complexity*, Vol. VI, Addison-Wesley pp. 1-47.

Laposky, B.F. 1975, Oscillons: Electronic Abstractions, in Leavitt, R. (ed) *Artist and Computer*, Harmony Books, New York, N.Y. pp. 21-22.

Laske, O. 1975, 'Introduction to a Generative Theory of Music', Sonological Reports, No. 1(B), Institute of Sonology, Utrecht.

Laske, O. 1979, 'Compositional Theory in Koenig's Project One and Project Two', *Computer Music Journal*, **5**(4), pp. 54-65.

Latour, B. 1979, *Laboratory Life, the Construction of Scientific Facts,* Princeton University Press, Princeton.

Leach, J. & J. Fitch 1995, 'Nature, Music, and Algorithmic Composition', *Computer Music Journal*, **19**(2), pp. 23-33.

Lerdhal, F. & R. Jackendoff 1983, *A Generative Theory of Tonal Music,* MIT Press, Cambridge, MA.

Levoy, M. 1990, 'Efficient Ray Tracing of Volume Data', *ACM Transactions on Graphics*, **9**(3), pp. 245-261.

Lewes, G.H. 1879, *Problems of Life and Mind,* Trubner, London.

Lidov, D. & J. Gabura 1973, 'A Melody-Writing Algorithm Using a Formal Language Model', *Computers and the Humanities*, **4**(3-4), pp. 138-148.

Lienhardt, P. 1988, 'Free-Form Surfaces Modeling by Evolution Simulation', *Proceedings of Eurographics '88*, Nice, France, pp. 327-341.

Lindenmayer, A. 1968, 'Mathematical Models for Cellular Interactions in Development, Parts I and II', *Journal of Theoretical Biology*, **18**, pp. 280-315.

Lindenmayer, A. 1974, Adding Continuous Components to L-Systems, in Rozenberg, G. & A. Salomaa (eds), *L Systems*, Lecture Notes in Computer Science, Vol. 15, Springer-Verlag, Berlin. pp. 53-68.

Lindenmayer, A. & G. Rozenberg (eds) 1976, *Automata, Languages, Development*, North-Holland, Amsterdam.

Lintermann, B. & O. Deussen 1996, 'Interactive Modelling of Branching Structures', *SIGGRAPH '96 (Technical Sketches)*, New Orleans.

Lintermann, B. & O. Deussen 1998, 'A Modelling Method and Interface for Creating Plants', *Computer Graphics Forum*, **17**(1), pp. 73-82.

Lintermann, B. & O. Deussen 1999, 'Interactive Modeling of Plants', *IEEE Computer Graphics & Applications*, **19**(1), pp. 2-11.

Little, T.D.C. & A. Ghafoor 1990, 'Synchronization and Storage Models for Multimedia Objects', *IEEE Journal on Selected Areas in Communications*, 8(3), pp. 413-427.

Lorenz, R. & I. Cunningham 1996, *Imogen Cunningham: Flora,* Little Brown and Company, Boston.

Loy, G. 1989, Composing with Computers – a Survey of Some Compositional Formalisms and Music Programming Languages, in Matthews, M.V. & J.R. Pierce (eds), *Current Directions in Computer Music Research*, MIT Press, Cambridge, MA. pp. 291-396.

Loy, G. & C. Abbott 1985, 'Programming Languages for Computer Music Synthesis, Performance and Composition', *ACM Computing Surveys*, **17**(2).

Lyon, D. 1995, 'Using Stochastic Petri Nets for Real-Time *N*th-Order Stochastic Composition', *Computer Music Journal*, **19**(4), pp. 13-22.

Lyotard, J.F. 1984, *The Postmodern Condition: A Report on Knowledge,* Theory and History of Literature, vol. 10, University of Minnesota Press, Minneapolis.

Lyotard, J.F. & A. Benjamin 1989, *The Lyotard Reader,* Blackwell, Oxford, UK; Cambridge, Mass., USA.

Macey, D. 2000, *The Penguin Dictionary of Critical Theory,* Penguin, London.

Machamer, P. 2002, A Brief Historical Introduction to the Philosophy of Science, in Machamer, P. & M. Silberstein (eds), *The Blackwell Guide to the Philosophy of Science*, Blackwell Philosophical Guides, Blackwell, Oxford, UK. pp. 1-17.

Magee, B. 1982, *Popper,* (10th impression, with revised postscript and bibliography), Fontana Modern Masters, Fontana, London.

Mandelbrot, B.B. 1983, *The Fractal Geometry of Nature,* (Updated and augmented edition), W.H. Freeman, New York.

Mandler, G. 1975, *Mind and Emotion,* Wiley, New York.

Mason, S. & M. Saffle 1994, 'L-Systems, Melodies and Musical Structure', *Leonardo Music Journal*, **4**, pp. 31-38.

Mastin, G.A., P.A. Watterberg & J.F. Mareda 1987, 'Fourier Synthesis of Ocean Scenes', *IEEE Computer Graphics and Applications*, (March), pp. 16-23.

Mathews, M.V. & F.R. Moore 1970, 'Groove: A Program to Compose, Store and Edit Functions of Time', *Communications of the ACM*, **13**(12), pp. 715-721.

Max, N.L. 1981, 'Vectorized Procedural Models for Natural Terrain: Waves and Islands in the Sunset', *Computer Graphics*, **15**(3), pp. 317-324.

McAlpine, K. 2000, *Applications of Dynamical Systems to Music Composition,* Ph.D. thesis, Department of Mathematics, University of Glasgow, Glasgow.

McAlpine, K., E.R. Miranda & S. Hoggar 1999, 'Making Music with Algorithms: A Case-Study System', *Computer Music Journal*, **23**(2), pp. 9-30.

McCormack, J. 1992a, *Flux* in ACM SIGGRAPH Video Review, ACM SIGGRAPH, New York.

McCormack, J. 1992b, *'Bloom' and 'Shell'* in ACM Siggraph Stereoscopic Slide Set, ACM SIGGRAPH, New York.

McCormack, J. 1993, Interactive Evolution of L-System Grammars for Computer Graphics Modelling, in Green, D. & T. Bossomaier (eds), *Complex Systems: From Biology to Computation*, ISO Press, Amsterdam. pp. 118-130.

McCormack, J. 1994a, *Turbulence: An Interactive Museum of Unnatural History*, Jon McCormack/Australian Film Commission, Melbourne.

McCormack, J. 1994b, Turbulence: An Interactive Installation Exploring Artificial Life. Visual Proceedings: The Art and Interdisciplinary Programs of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics* Annual Conference Series, ACM SIGGRAPH, New York, pp. 182-183.

McCormack, J. 1994c, Wild: An Interactive Computer Installation, in Sproul, L. (ed) *The 1994 Next Wave Art and Technology Catalogue*, Next Wave Festival, Inc., Melbourne. pp. 22-25.

McCormack, J. 1996, Grammar-Based Music Composition, in Stocker, R., H. Jelinek, B. Durnota & T. Bossomaier (eds), *Complex Systems 96: From Local Interactions to Global Phenomena*, ISO Press, Amsterdam. pp. 321-336.

McCormack, J. 2002, 'Evolving for the Audience', *International Journal of Design Computing*, **4**(Special Issue on Designing Virtual Worlds).

McCormack, J. & A. Sherstyuk 1997, 'Creating and Rendering Convolution Surfaces', Technical Report, No. 97/324, October 1997. Monash University, Melbourne.

McCormack, J. & A. Sherstyuk 1998, 'Creating and Rendering Convolution Surfaces', *Computer Graphics Forum*, **17**(2), pp. 113-121.

McCormack, J. & A. Dorin 2001, 'Art, Emergence and the Computational Sublime' in Dorin, A. (ed), *Second Iteration: a conference on generative systems in the electronic arts*, CEMA, Melbourne, Australia, pp. 67-81.

McDonough, R. 2002, Emergence and Creativity: Five Degrees of Freedom, in Dartnall, T. (ed) *Creativity, Cognition, and Knowledge*, Perspectives on Cognitive Science, Praeger, Westport, Connecticut; London. pp. 283-320.

McKenna, M. & D. Zeltzer 1990, Dynamic Simulation of Autonomous Legged Locomotion. Proceedings of SIGGRAPH '90 (Dallas, Texas, August 6-10, 1990). In *Computer Graphics* **24**(4) ACM SIGGRAPH, New York, pp. 29-38.

McLaughlin, B.P. 2001, Emergentism, in Wilson, R.A. & F.C. Keil (eds), *The MIT Encyclopedia of the Cognitive Sciences*, The MIT Press, Cambridge, Massachusetts. pp. 267-268.

Mech, R. & P. Prusinkiewicz 1996, Visual Models of Plants Interacting with Their Environment. Proceedings of SIGGRAPH 96 (New Orleans, Louisiana, August 4-9, 1996). In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 397-410.

Mill, J.S. 1872, *A System of Logic, Ratiocinative and Inductive: Being a Connected View of the Principles of Evidence and the Methods of Scientific Investigation,* (8th Edition), Longman, London.

Miller, G.F. 2000, *The Mating Mind: How Sexual Choice Shaped the Evolution of Human Nature,* William Heinemann, London.

Miller, G.S.P. 1988, The Motion Dynamics of Snakes and Worms. Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1998). In *Computer Graphics* **22**(4) ACM SIGGRAPH, New York, pp. 169-178.

Minsky, M. (ed.) 1965, *Semantic Information Processing*, MIT Press, Cambridge, MA.

Minsky, M. 1981, 'Music, Mind and Meaning', *Computer Music Journal*, **5**(3), pp. 28-44.

Miranda, E.R. 2001, *Composing Music with Computers,* Music Technology Series, Focal Press, Oxford ; Boston.

Mitchell, M. 1996, *Introduction to Genetic Algorithms,* Complex Adaptive Systems, MIT Press, Cambridge, MA.

Mitter, P. 1999, 'A Short Commentary of 'the Science of Art'', *Journal of Consciousness Studies*, **6**(6-7), pp. 64-65.

Miyata, H. 1986, 'Finite-Difference Simulation of Breaking Waves', *Journal of Computational Physics*, **65**, pp. 179-214.

Mock, K.J. 1998, 'Wildwood: The Evolution of L-Systems Plants for Virtual Environments', *International Conference on Evolutionary Computing '98*, vol. Proceedings of the 1998 IEEE World Congress on Computational Intelligence, IEEE-Press, Anchorage, Alaska, pp. 476-480.

Moholy-Nagy, L. 1967, 'A New Instrument of Vision in "from the Bauhaus"', *Camera*, **46**(4), p. 30.

Molino, J. 1975, 'Fait Musicale Et Sémiologies De La Musique', *Musique en jeu*, **17**, pp. 37-62.

Monod, J. 1971, *Chance and Necessity; an Essay on the Natural Philosophy of Modern Biology,* Penguin, London.

Moore, F.R. 1990, *Elements of Computer Music,* Prentice Hall, Englewood Cliffs, N.J.

Moravac, H. 1988, *Mind Children: The Future of Robot and Human Intelligence,* Harvard University Press, Cambridge.

Morgan, C.L. 1923, *Emergent Evolution: The Gifford Lectures. 1923,* Williams and Norgate, London.

Murray, C.D. 1927, 'A Relationship between Circumstance and Weight in Trees and Its Bearing on Branching Angles', *Journal of General Physiology*, **10**, pp. 725-729.

Musgrave, F.K. 1994, Procedural Fractal Terrains, in Ebert, D.S. (ed) *Texturing and Modeling: A Procedural Approach*, Academic Press, London. pp. 295-310.

Musgrave, F.K., C.E. Kolb & R.S. Mace 1989, The Synthesis and Rendering of Eroded Fractal Terrains. SIGGRAPH '89 Conference Proceedings (Boston, Massachusetts, 31 July–4August). In *Computer Graphics* **23**(3) pp. 41-50.

Nagel, E. 1961, *The Structure of Science: Problems in the Logic of Scientific Explanation,* Routledge, London.

Nake, F. 1998, 'Art in the Time of the Artificial', *Leonardo*, **31**(3), pp. 163-164.

Nake, F. 2002, 'About Generative Aesthetics [Email]', Personal Communication.

Nelson, G.L. 1996, 'Real Time Transformation of Musical Material with Fractal Algorithms', *Computers & Mathematics with Applications*, **32**(1), pp. 109-116.

Nevill-Manning, C.G. & I.H. Witten 1997, 'Compression and Explanation Using Hierarchical Grammars', *The Computer Journal*, **40**(2/3), pp. 103-116.

Nguyen, D.C. 1997*, Ray Traced Evolution – User's Manual* (Web Page), <http://www.rz.tu-ilmenau.de/~juhu/GX/RTEvol/DOC/LATEX2HTML/manual.html> (Accessed 11 May 2002).

Niklas, K.J. 1982, 'Computer Simulations of Early Land Plant Branching Morphologies: Canalization of Patterns During Evolution?' *Paleobiology*, 8(3), pp. 196-210.

Niklas, K.J. 1986, 'Computer Simulated Plant Evolution', *Scientific American*, pp. 55-64.

Niklas, K.J. 1997, *The Evolutionary Biology of Plants,* University of Chicago Press, Chicago, IL.

Nishimura, H., H. Ohno, T. Kawata, I. Shirakawa & K. Omura 1983, Links-1: A Parallel Pipelined Multimicrocomputer System for Image Creation, in *Conference Proceedings of the 10th Annual International Symposium on Computer Architecture, Sigarch*, pp. 387-394.

Nishimura, H., M. Hirai, T. Kawai, T. Kawata, I. Shirakawa & K. Omura 1985, 'Object Modelling by Distribution Function and a Method of Image Generation', *Transactions of the Institute of Electronics and Communication Engineers of Japan*, **J68-D**(4), pp. 718-725.

Norman, D.A. & S.W. Draper 1986, *User Centered System Design : New Perspectives on Human-Computer Interaction,* Lawrence Erlbaum Associates, Hillsdale, N.J.

Noser, H. & D. Thalmann 1993, 'L-System Based Behavioural Animation', *Pacific Graphics '93*, World Scientific, Seoul, Korea, pp. 133-146.

Noser, H., D. Thalmann & R. Turner 1992, 'Animation Based on the Interaction of L-Systems with Vector Force Fields', *Computer Graphics International*, Springer-Verlag, pp. 747-761.

Nyman, M. 1999, *Experimental Music - Cage and Beyond,* (Second Edition), Music in the 20th Century, Cambridge University Press, Cambridge.

Ochoa, G. 1998, 'On Genetic Algorithms and Lindenmayer Systems', *PPSN IV*, vol. LNCS 1498, Springer-Verlag, Amsterdam, pp. 335-343.

Omura, K., Y. Kawaguchi & S. Noji 1985, *CG in Japan,* Graphic-sha Publishing, Tokyo.

Oppenheimer, P.E. 1986, Real Time Design and Animation of Fractal Plants and Trees. Proceedings of SIGGRAPH '86 (Dallas, Texas, August 18-22, 1986, Evans, D.C. & R.J. Athay, eds). In *Computer Graphics* **20**(4) ACM SIGGRAPH, New York, pp. 55-64.

Oppenheimer, P.E. 1988, The Artificial Menagerie, in Langton, C.G. (ed) *Artificial Life, SFI Studies in the Sciences of Complexity*, Addison-Wesley, Redwood City, California. pp. 251-274.

Oreskes, N., K. Shrader-Frechette & K. Belitz 1994, 'Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences', *Science*, **263**(4 February 1994), pp. 641-646.

Orff, C. 1967, *Gessprâche Mit Komponisten,* Zürich.

Parish, Y.I.H. & P. Müller 2001, Procedural Modeling of Cities. Proceedings of SIGGRAPH 2001 (Los Angeles, California, August 12-17). In *Computer Graphics Proceedings* Annual Conference Series, ACM SIGGRAPH, pp. 301-308.

Pattee, H.H. 1988, Simulations, Realizations, and Theories of Life, in Langton, C.G. (ed) *Artificial Life*, SFI Studies in the Sciences of Complexity, Vol. VI, Addison-Wesley pp. 63-77.

Paul, C. 2003, *Digital Art,* Thames & Hudson World of Art, Thames and Hudson, London.

Peachey, D.R. 1985, Solid Texturing of Complex Surfaces. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22-26, 1985, Barsky, B.A., ed). In *Computer Graphics* **19**(3) ACM SIGGRAPH, New York, pp. 279-286.

Peachey, D.R. 1986, Modeling Waves and Surf. Proceedings of SIGGRAPH '86 (Dallas, Texas, August 18-22, 1986, Evans, D.C. & R.J. Athay, eds). In *Computer Graphics* **20**(4) ACM SIGGRAPH, New York, pp. 65-74.

Peachey, D.R. 1994, Building Procedural Textures, in Ebert, D.S. (ed) *Texturing and Modeling: A Procedural Approach*, Academic Press, London. pp. 5-100.

Peitgen, H.-O. & P.H. Richter 1986, *The Beauty of Fractals: Images of Complex Dynamical Systems,* Springer-Verlag, Berlin; New York.

Peitgen, H.-O. & D. Saupe 1988, *The Science of Fractal Images,* Springer-Verlag, Berlin; New York.

Penny, S. 1996, Artistic Practice, Body Knowledge and the Engineering World View, in Stocker, G. & C. Schöpf (eds), *Ars Electronica '96: Memesis*, Springer-Verlag, Wein. pp. 190-207.

Penny, S. 1999, 'Systems Aesthetics and Cyborg Art: The Legacy of Jack Burnham', *Sculpture*, **18**(1), pp. 36-41.

Penrose, R. 1989, *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics,* Oxford University Press, Oxford, England.

Perlin, K. 1985a, An Image Synthesizer. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22-26, 1985, Barsky, B.A., ed). In *Computer Graphics* **19**(3) ACM SIGGRAPH, New York, pp. 287-296.

Perlin, K. 1985b, 'An Image Synthesizer', *Computer Graphics*, **19**(3), pp. 287-296.

Phong, B.T. 1975, 'Illumination for Computer Generated Pictures', *Communications of the ACM*, **18**(6), pp. 311-317.

Pinker, S. 1997, *How the Mind Works,* Penguin Press, Middlesex, England.

Poincaré, H. 1923, *The Foundations of Science: Science and Hypothesis, the Value of Science, Science and Method,* The Science Press, New York.

Polanyi, M. 1968, 'Life's Irreducible Structure', *Science*, **160**, pp. 1308-1312.

Pope, S. 1986, 'Music Notations and the Representation of Musical Structure and Knowledge', *Perspectives of New Music*, **24**(2), pp. 156-189.

Popper, K.R. 1968, *The Logic of Scientific Discovery,* (3rd edition), Hutchinson, London.

Praehofer, H. 1991, 'System Theoretic Formalisms for Combined Discrete-Continuous System Simulation', *International Journal of General Systems*, **19**(3), pp. 219-240.

Press, W.H., S.A. Teukolsky, W.T. Vetterling & B.P. Flannery 1992, *Numerical Recipes in C: The Art of Scientific Computing,* (Second Edition), Cambridge University Press, Cambridge.

Prigogine, I. & I. Stengers 1984, *Order out of Chaos: Man's New Dialogue with Nature,* (1st Edition), New Science Library: Distributed by Random House, Boulder, CO.

Prusinkiewicz, P. 1986a, 'Score Generation with L-Systems', *Proceedings of the 1986 International Computer Music Conference*, Computer Music Association, San Francisco, CA, pp. 455-457.

Prusinkiewicz, P. 1986b, 'Graphical Applications of L-Systems', *Proceedings of Graphics Interface '86 — Vision Interface*, CIPS, pp. 247-253.

Prusinkiewicz, P. 1987, Applications of L-Systems to Computer Imagery, in Ehrig, H., M. Nagl, A. Rosenfeld & G. Rozenberg (eds), *Graph Grammars and Their Application to Computer Science; Third International Workshop*, Lecture Notes in Computer Science, Vol. 291, Springer-Verlag, Berlin. pp. 534-548.

Prusinkiewicz, P. & J. Hanan 1989, *Lindenmayer Systems, Fractals and Plants,* Lecture Notes in Bio-Mathematics, vol. 79, Springer-Verlag, Berlin.

Prusinkiewicz, P. & A. Lindenmayer 1990, *The Algorithmic Beauty of Plants,* The Virtual Laboratory, Springer-Verlag, New York.

Prusinkiewicz, P. & J. Hanan 1990, Visualization of Botanical Structures and Processes Using Parametric L-Systems, in Thalmann, D. (ed) *Scientific Visualization and Graphics Simulation*, John Wiley & Sons, Chichester. pp. 183-201.

Prusinkiewicz, P. & M. Hammel 1993, 'A Fractal Model of Mountains with Rivers', *Graphics Interface '93*, Toronto, Ontario, pp. 174-180.

Prusinkiewicz, P., A. Lindenmayer & J. Hanan 1988, Developmental Models of Herbaceous Plants for Computer Imagery Purposes. Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988). In *Computer Graphics* **22**(4) ACM SIGGRAPH, New York, pp. 141-150.

Prusinkiewicz, P., M. Hammel & E. Mjolsness 1993, Animation of Plant Development. Proceedings of SIGGRAPH 93 (Anaheim, California, August 1-6, 1993). In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, New York, pp. 351-360.

Prusinkiewicz, P., M. James & R. Mech 1994, Synthetic Topiary. Proceedings of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 351-358.

Prusinkiewicz, P., J. Hanan & R. Mech 2000, An L-System-Based Plant Modeling Language, in Nagl, M., A. Schürr & M. Münch (eds), *Lecture Notes in Computer Science*, Vol. 1779, Springer-Verlag, Berlin. pp. 395-410.

Prusinkiewicz, P., M. Hammel, R. Mech & J. Hanan 1995, The Artificial Life of Plants. SIGGRAPH '95 Course Notes). In *Artificial Life for Graphics, Animation, and Virtual Reality* **7** ACM SIGGRAPH, pp. 1-1 - 1-38.

Prusinkiewicz, P., R. Karwowski, R. Mech & J. Hanan 2000, L-Studio/Cpfg: A Software System for Modelling Plants, in Nagl, M., A. Schürr & M. Münch (eds), *Lecture Notes in Computer Science*, Vol. 1779, Springer-Verlag, Berlin. pp. 457-464.

Prusinkiewicz, P., L. Mündermann, R. Karwowski & B. Lane 2001, The Use of Positional Information in the Modeling of Plants. Proceedings of SIGGRAPH 2001 (Los Angeles, California, August 12-17). In *Computer Graphics Proceedings* Annual Conference Series, ACM SIGGRAPH, pp. 289-300.

Putman, H. 1975, *Mathematics, Matter and Method,* vol. 1, Cambridge University Press, Cambridge.

Qazi, N.U., M. Woo & A. Ghafoor 1993, 'A Synchronization and Communication Model for Distributed Multimedia Objects', *ACM Multimedia 93*, ACM, Anaheim, CA, pp. 147-155.

Quammen, D. 1996, *Song of the Dodo: Island Biogeography in an Age of Extinctions,* Scribner, New York, NY.

Raibert, M.H. & J.K. Hodgins 1991, Animation of Dynamic Legged Locomotion. Proceedings of SIGGRAPH '91 (Las Vegas, Nevada, July 28 - August 2, 1991). In *Computer Graphics* **25**(4) ACM SIGGRAPH, New York, pp. 349-358.

Ramachandran, V.S. & W. Hirstein 1999, 'A Neurological Theory of Aesthetic Experience', *Journal of Consciousness Studies*, **6**(6-7), pp. 15-51.

Rawlins, G.J.E. (ed.) 1991, *Foundations of Genetic Algorithms*, Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Morgan Kaufmann Publishers, San Mateo, Calif.

Reeves, W.T. 1983, Particle Systems — a Technique for Modeling a Class of Fuzzy Objects. Proceedings of SIGGRAPH '83 (Detroit, Michigan, July 25-29, 1983, Tanner, P., ed). In *Computer Graphics* **17**(3) ACM SIGGRAPH, New York, pp. 359-376.

Reeves, W.T. & R. Blau 1985, Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems. Proceedings of SIGGRAPH '85 (San Francisco, California, July 22-26, 1985, Barsky, B.A., ed). In *Computer Graphics* **19**(3) ACM SIGGRAPH, New York, pp. 313-322.

Reichardt, J. 1971, *The Computer in Art,* Studio Vista; Van Nostrand Reinhold, London; New York.

Reynolds, C.W. 1987, Flocks, Herds, Schools: A Distributed Behavioral Model. Proceedings of SIGGRAPH '87 (Anaheim, California, July 27-31, 1987, Stone, M.C., ed). In *Computer Graphics* **21**(4) ACM SIGGRAPH, New York**,** pp. 25-34.

Ridley, J.N. 1986, 'Ideal Phyllotaxis on General Surfaces of Revolution', *Mathematical Biosciences*, **79**, pp. 1-24.

Ripley, B.D. 1977, 'Modelling Spatial Patterns', *Journal of the Royal Statistical Society. Series B (Methodological)*, **39**(2), pp. 172-212.

Risan, L.C. 1997*, Artificial Life: A Technoscience Leaving Modernity? An Anthropology of Subjects and Objects*, <http://www.anthrobase.com/txt/Risan_L_05.htm> (Accessed 13 August 2001).

Roads, C. 1978, *Composing Grammars,* International Computer Music Association, San Francisco, CA.

Roads, C. 1985, Grammars as Representations for Music, in Roads, C. & J. Strawn (eds), *Foundations of Computer Music*, MIT Press, Cambridge, MA. pp. 403-442.

Roads, C. 1996, *The Computer Music Tutorial,* MIT Press, Cambridge, Mass.

Rooke, S. 2002, Eons of Genetically Evolved Algorithmic Images, in Bentley, P.J. & D.W. Corne (eds), *Creative Evolutionary Systems*, Academic Press, London. pp. 339-365.

Rosenberg, M.J. 1983, *The Cybernetics of Art: Reason and the Rainbow,* Studies in Cybernetics, vol. 4, Gordon and Breach Science, New York.

Rosenman, M.A. 1997, The Generation of Form Using an Evolutionary Approach, in Dasgupta, D. & Z. Michalewicz (eds), *Evolutionary Algorithms in Engineering Applications*, Springer-Verlag, Southampton and Berlin. pp. 69-85.

Rothstein, J. 1992, *MIDI : A Comprehensive Introduction,* Oxford University Press, Oxford.

Rozenberg, G. & A. Salomaa 1980, *The Mathematical Theory of L-Systems,* Academic Press, New York.

Rubin, E. 1921, *Visuell Wahrgenommene Figuren,* Glydendalske, Copenhagen.

Runqiang, B., P. Chen, K. Burrage, J. Hanan, P.M. Room & J. Belward 2002, Derivation of L-System Models from Measurements of Biological Branching Structures Using Genetic Algorithms, in Hendtlass, T. & M. Ali (eds), *IEA/AIE 2002, Lecture Notes in Artificial Intelligence 2358*, Springer-Verlag, Berlin.

Salomaa, A. 1973, *Formal Languages,* Academic Press, New York, NY.

Saunders, R. 1999*, A Computational Model of Simple Creativity Using Emergence* (web page), <http://www.arch.usyd.edu.au/~rob/study/publications/proposal/1999SaundersThesisProposal.html> (Accessed 21 November 2001).

Scarborough, D., B. Miller & J. Jones 1989, 'Connectionist Models for Tonal Analysis', *Computer Music Journal*, **13**(3), pp. 49-55.

Schillinger, J. 1948, *The Mathematical Basis of the Arts,* The Philosophical Library, New York.

Schopenhauer, A. 1928, *The Works of Schopenhauer,* (With an Introduction by Thomas Mann), Frederick Ungar Publishing Co., New York.

Sebeok, T. 1975, 'Six Species of Signs: Some Propositions and Strictures', *Semiotica*, **13**(3), pp. 233-260.

Selfridge-Field, E. (ed.) 1997, *Beyond MIDI: The Handbook of Musical Codes*, MIT Press, Cambridge, Massachusetts.

Shampine, L.F., I. Gladwell & R.W. Brankin 1991, 'Reliable Solution of Special Event Location Problems for ODEs', *ACM Transactions on Mathematical Software*, **17**(1), pp. 11-25.

Shapin, S., T. Hobbes & S. Schaffer 1985, *Leviathan and the Air-Pump: Hobbes, Boyle, and the Experimental Life: Including a Translation of Thomas Hobbes, Dialogus Physicus De Natura Aeris by Simon Schaffer,* Princeton University Press, Princeton, N.J.

Sharp, D.H. 1998, *LMUSe Software Web Site* (Web Page), <http://www.geocities.com/Athens/Academy/8764/lmuse/lmuse.html> (Accessed 3 August 2002).

Simon, H.A. 1996, *The Sciences of the Artificial,* (3rd Edition), MIT Press, Cambridge, Mass.

Sims, K. 1990a, *Panspermia* in SIGGRAPH Video Review, ACM SIGGRAPH, New York.

Sims, K. 1990b, Particle Animation and Rendering Using Data Parallel Computation. Proceedings of SIGGRAPH '90 (Dallas, Texas, August 6-10, 1990). In *Computer Graphics* **24**(4) ACM SIGGRAPH, New York, pp. 405-413.

Sims, K. 1991a, 'Interactive Evolution of Dynamical Systems', *First European Conference on Artificial Life*, MIT Press, Paris, pp. 171-178.

Sims, K. 1991b, Artificial Evolution for Computer Graphics. Proceedings of SIGGRAPH '91 (Las Vegas, Nevada, July 28 - August 2, 1991). In *Computer Graphics* **25**(4) ACM SIGGRAPH, New York, pp. 319-328.

Sims, K. 1993, 'Interactive Evolution of Equations for Procedural Models', *The Visual Computer*, **9**, pp. 466-476.

Sims, K. 1994a, 'Evolving 3D Morphology and Behavior by Competition' in Brooks, R. & P. Maes (eds), *Proceedings of Artificial Life IV*, MIT Press, pp. 28-39.

Sims, K. 1994b, Evolving Virtual Creatures. Proceedings of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 15-22.

Smart, J.J.C. 1963, *Philosophy and Scientific Realism,* Routledge and Kegan Paul, London.

Smith, A.R. 1984, Plants, Fractals and Formal Languages. Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July 23-27, 1984, Christiansen, H., ed). In *Computer Graphics* **18**(3) Proceedings of SIGGRAPH '84, Minneapolis, Minnesota, July 22-27, ACM SIGGRAPH, New York, pp. 1-10.

Snow, C.P. 1959, *The Two Cultures and the Scientific Revolution,* Rede Lectures: 1959, Cambridge University Press, London.

Soddell, F. & J. Soddell 2000, 'Microbes and Music' in Mizoguchi, R. & J.K. Slaney (eds), *PRICAI 2000, Topics in Artificial Intelligence, The Sixth Pacific Rim International Conference on Artificial Intelligence*, vol. 1886, Springer-Verlag, Melbourne, Australia, pp. 767-777.

Soddu, C. 1998, 'Argenia, a Natural Generative Design' in Soddu, C. (ed), *Generative Art '98*, Milano, Italy.

Sokal, A. 1996a, 'Transgressing the Boundaries: Toward a Transformative Hermeneutics of Quantum Gravity', *Social Text*, **46/47**, pp. 217-252.

Sokal, A. 1996b, 'A Physicist Experiments with Cultural Studies', *Lingua Franca*, **May/June**, pp. 62-64.

Sommerer, C. & L. Mignonneau 1998, Art as a Living System, in Sommerer, C. & L. Mignonneau (eds), *Art@Science*, Springer, Wein. pp. 148-161.

Soper, K. 1995, *What Is Nature? Culture, Politics and the Non-Human,* Blackwell Publishers Ltd, Oxford, UK.

Stam, J. & E. Fiume 1995, Depicting Fire and Other Gaseous Phenomena Using Diffusion Processes. Proceedings of SIGGRAPH 95 (Los Angeles, California, August 6-11, 1995). In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 129-136.

Stevens, P.S. 1974, *Patterns in Nature,* Little Brown, Boston, Mass.

Stiny, G. 1975, *Pictorial and Formal Aspects of Shape and Shape Grammars,* Isr, Interdisciplinary Systems Research ; 13., Birkhäuser, Basel ; Stuttgart.

Stiny, G. & J. Gips 1978, *Algorithmic Aesthetics: Computer Models for Criticism and Design in the Arts,* University of California Press, Berkeley; Los Angeles, CA.

Stocker, G. & C. Schöpf (eds) 2001, *Ars Electronica 2001: Takeover - Who's Doing the Art of Tomorrow?*, Springer, Wein.

Supper, M. 2001, 'A Few Remarks on Algorithmic Composition', *Computer Music Journal*, **25**(1), pp. 48-53.

Suppes, P. 1960, 'A Comparison of the Meaning and Uses of Models in Mathematics and the Empirical Sciences', *Synthese*, **12**, pp. 287-301.

Szilard, A.L. & R.E. Quinton 1979, 'An Interpretation for DOL System by Computer Graphics', *The Science Terrapin*, **4**, pp. 8-13.

Tabuada, P., P. Alves, P. Gomes & P. Rosa 1998, '3D Artificial Art by Genetic Algorithms', *Workshop on Evolutionary Design at Artificial Intelligence in Design - AID' 98*, pp. 18-21.

Tasi´c, V. 2001, *Mathematics and the Roots of Postmodern Thought,* Oxford University Press, Oxford.

Taylor, T. 2002, Creativity in Evolution: Individuals, Interactions, and Environments, in Bentley, P.J. & D.W. Corne (eds), *Creative Evolutionary Systems*, Academic Press, London. pp. 79-108.

Thom, R. 1975, *Structural Stability and Morphogenesis: An Outline of a General Theory of Models,* (1st ed. English), W. A. Benjamin, Reading, Mass.

Thompson, D.A.W. 1942, *On Growth and Form,* (2nd Edition), Cambridge University Press, Cambridge.

Thompson, D.A.W. 1961, *On Growth and Form,* (Abridged Edition), Cambridge University Press, Cambridge.

Todd, P.M. & D.G. Loy 1991, *Music and Connectionism,* MIT Press, Cambridge, Mass.

Todd, P.M. & G.M. Werner 1998, Frankensteinian Methods for Evolutionary Music Composition, in Griffith, N. & P.M. Todd (eds), *Musical Networks: Parallel Distributed Perception and Performance*, MIT Press/Bradford Books, Cambridge, MA.

Todd, S. & W. Latham 1991, 'Mutator: A Subjective Human Interface for Evolution of Computer Sculptures', Technical Report, No. 248, IBM United Kingdom Scientific Centre.

Todd, S. & W. Latham 1992, *Evolutionary Art and Computers,* Academic Press, London.

Tonkin, J. 2001*, John Tonkin's Web Site* (web page), <http://www.johnt.org> (Accessed October 10 2001).

Towner, G. 1999, *Discovering Quicktime: An Introduction for Windows and Macintosh Programmers,* Morgan Kaufmann, San Diego, CA.

Traxler, C. & M. Gervautz 1996, 'Using Genetic Algorithms to Improve the Visual Quality of Fractal Plants Generated with CSG-Pl-Systems', Research Report, No. TR-186-2-96-04, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria.

Tu, X. & D. Terzopoulos 1994, Artificial Fishes: Physics, Locomotion, Perception, Behavior. Proceedings of SIGGRAPH 94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 43-50.

Turing, A.M. 1952, 'The Chemical Basis of Morphogenesis', *Philosophical Transactions of the Royal Society*, **237**, pp. 37-72.

Turk, G. 1991, Generating Textures for Arbitrary Surfaces Using Reaction-Diffusion. Proceedings of SIGGRAPH '91 (Las Vegas, Nevada, July 28 - August 2, 1991). In *Computer Graphics* **25**(4) ACM SIGGRAPH, New York, pp. 289-298.

Tymoczko, T. 1986, *New Directions in the Philosophy of Mathematics: An Anthology,* Birkhäuser, Boston.

Upstil, S. 1990, *The Renderman Companion, a Programmer's Guide to Realistic Computer Graphics,* Addison-Wesley, Reading, Massachusetts.

Van Fraassen, B.C. 1980, *The Scientific Image,* Clarendon Library of Logic and Philosophy, Clarendon Press; Oxford University Press, Oxford; New York.

Ventrella, J. 1995a, 'Eukaryotic Virtual Reality', *ISEA '95: International Symposium on Electronic Art*, Montréal, Canada, p. 10.

Ventrella, J. 1995b, 'Disney Meets Darwin — the Evolution of Funny Animated Figures' in Thalmann, N.M. & D. Thalmann (eds), *Computer Animation '95*, IEEE, Montréal, pp. 35-43.

Vogel, H. 1979, 'A Better Way to Construct the Sunflower Head', *Mathematical Biosciences*, **44**, pp. 179-189.

Voss, R.F. & J. Clarke 1975, '1/F Noise in Music and Speech', *Nature*, **258**, pp. 317-318.

Wainwright, S.A. 1988, *Axis and Circumference: The Cylindrical Shape of Plants and Animals,* Harvard University Press, Cambridge, Mass.

Wallace, A.R. 1855, 'On the Law Which Has Regulated the Introduction of New Species', *Annals and Magazine of Natural History*, **16**(September 1855).

Waller, D.M. & D.A. Steingraeber 1985, Branching and Molecular Growth: Theoretical Models and Empirical Patterns, in Jackson, J.B.C., L.W. Buss & R.E. Cook (eds), *Population Biology and Evolution of Clonal Organisms*, Yale University Press pp. 225-257.

Watt, A. & M. Watt 1993, *Advanced Animation and Rendering Techniques: Theory and Practice,* Addison-Wesley, New York.

Weber, M. & S.N. Eisenstadt 1968, *Max Weber on Charisma and Institution Building; Selected Papers,* The Heritage of Sociology, University of Chicago Press, Chicago.

Wertheimer, M. 1938, Laws of Organization in Perceptual Forms, in Ellis, W.D. (ed) *A Sourcebook of Gestalt Psychology*, Harcourt Brace, New York. pp. 71-88.

Weston, E. 1973, *The Daybooks of Edward Weston,* Aperture, Millerton, N.Y.

Whitelaw, M. 1998, *Rethinking a Systems Aesthetic* (MS Word file), <http://comedu.canberra.edu.au/~mitchellw/papers/systems.msw.hqx> (Accessed 10 October 2001).

Whitelaw, M. 1999, 'The Abstract Organism: Towards a Prehistory for a-Life Art' in Dorin, A. & J. McCormack (eds), *First Iteration: a conference on generative systems in the electronic arts*, CEMA, Melbourne, Australia, pp. 176-184.

Whitelaw, M. 2000, *Artificial Life in New Media Art,* Ph.D. thesis, UTS, Sydney.

Whitelaw, M. 2002, Breeding Aesthetic Objects: Art and Artificial Evolution, in Bentley, P.J. & D.W. Corne (eds), *Creative Evolutionary Systems*, Academic Press, London. pp. 129-145.

Whitted, T. 1980, 'An Improved Illumination Model for Shaded Display', *Communications of the ACM*, **23**(6), pp. 343-349.

Wiener, N. 1961, *Cybernetics: Or Control and Communication in the Animal and the Machine,* (2nd Edition), MIT Press, Cambridge, Mass.

Wiggins, G., G. Papadopoulos, S. Phon-Amnuaisuk & A. Tuson 1999, 'Evolutionary Methods for Musical Composition', *Proceedings of the CASYS98 Workshop on Anticipation, Music & Cognition*.

Williams, L. 1983, Pyramidal Parametrics. Proceedings of SIGGRAPH '83 (Detroit, Michigan, July 25-29, 1983, Christiansen, H., ed). In *Computer Graphics* **17**(3) ACM SIGGRAPH, New York, pp. 1-11.

Wilson, R.A. & F.C. Keil (eds) 1999, *The MIT Encyclopaedia of the Cognitive Sciences*, MIT Press, Cambridge, Massachusetts.

Wilson, S. 2002, *Information Arts: A Survey of Art and Research at the Intersection of Art, Science, and Technology,* Leonardo, MIT Press, Cambridge, Mass.

Wimsatt, W. 1980, 'Randomness and Perceived-Randomness in Evolutionary Biology', *Synthese*, **43**, pp. 287-329.

Winograd, T. 1968, 'Linguistics and the Computer Analysis of Tonal Harmony', *Journal of Music Theory*, **12**(1), pp. 2-49.

Wirfs-Brock, R., B. Wilkerson & L. Wiener 1990, *Designing Object-Oriented Software,* Prentice Hall, Englewood Cliffs, N.J.

Wishart, T. 1996, *On Sonic Art,* (New and Revised Edition edited by Simon Emmerson), Contemporary Music Series, vol. 12, Harwood Academic Publishers GmbH, Amsterdam.

Witkin, A. & M. Kass 1991, Reaction-Diffusion Textures. Proceedings of SIGGRAPH '91 (Las Vegas, Nevada, July 28 - August 2, 1991). In *Computer Graphics* **25**(4) ACM SIGGRAPH, New York, pp. 299-308.

Wittgenstein, L. & G.H.v. Wright 1980, *Culture and Value,* (2nd edition), B. Blackwell, Oxford.

Wolfe, T. 1975, *The Painted Word,* Farrar Straus and Giroux, New York.

Wolpert, L. 1993, *The Unnatural Nature of Science,* Faber and Faber Limited, London.

Woo, M., J. Neider, T. Davis & D. Shreiner 1997, *The Open GL Programming Guide,* (Third Edition), Addison-Wesley, Boston.

Worrall, J. 2002, Philosophy of Science: Classic Debates, Standard Problems, Future Prospects, in Machamer, P. & M. Silberstein (eds), *The Blackwell Guide to the Philosophy of Science*, Blackwell Philosophical Guides, Blackwell, Oxford, UK. pp. 18-36.

Xenakis, I. 1960, 'Elements of Stochastic Music', *Gravesaner Blätter*, **18**, pp. 84-105.

Xenakis, I. 1971, *Formalized Music: Thought and Mathematics in Composition,* Indiana University Press.

Xenakis, I. 1992, *Formalized Music,* (Revised Edition), Pendragan Press, New York.

Yokomori, T. 1980, 'Stochastic Characterizations of EOL Languages', *Information and Control*, **45**, pp. 26-33.

Zabusky, N.J. 1984, 'Computational Synergetics', *Physics Today*, (July), pp. 36-46.

Zeigler, B.P., T.G. Kim & H. Praehofer 2000, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems,* (2nd edition), Academic, San Diego, California; London.

Zhi, W., Z. Ming & Y. Qi-Xing 2001, 'Modeling of Branching Structures of Plants', *Journal of Theoretical Biology*, **209**, pp. 383-394.